futuredata

FUTUREDATA TAPE OPERATING SYSTEM

USERS MANUAL

MICROCOMPUTER
SYSTEMS

Received 11/14/78

FUTUREDATA TAPE OPERATING SYSTEM

USERS MANUAL

FUTUREDATA COMPUTER CORP.
11205 S. La Cienega Blvd.
Los Angeles, CA 90045

Copyright 1977, FUTUREDATA COMPUTER CORP.

TABLE OF CONTENTS

	Page
1.0 Microsystem Overview	1-1
1.1 Hardware Features	1-1
1.1.1 CPU	1-1
1.1.1.1 8080A CPU	1-1
1.1.1.2 6800 CPU	1-2
1.1.1.3 Z-80 CPU	1-2
1.1.2 Processor Independent System Bus	1-2
1.1.3 Universal Bus Configuration	1-3
1.1.4 Crystal-Controlled Processor Clock	1-3
1.1.5 Bootstrap Loader in PROM	1-3
1.1.6 System Reset	1-4
1.1.7 Power-On Bootstrap	1-4
1.1.8 Direct Memory Access	1-4
1.1.9 Priority Interrupts	1-4
1.1.9.1 8-Level Interrupts for the 8080A and Z-80	1-4
1.1.9.2 4-Level Interrupt for the 6800	1-4
1.1.10 System Interrupt Mask	1-4
1.1.11 I/O Device Interrupt Masks	1-5
1.1.12 8K Memory Modules	1-5
1.1.13 Memory Write-Protect	1-5
1.1.14 CRT Display	1-5
1.1.15 CRT Display Memory Page Select	1-6
1.1.16 53-Key ASCII Keyboard	1-6
1.1.17 Breakpoint Interrupt	1-6
1.1.18 Load, Reset Buttons and Status Display	1-6
1.1.19 Cassette Tape Units	1-7
1.1.20 EIA Interfaces	1-7
1.1.21 Real-Time Clock	1-7
1.1.22 8-Bit I/O Port	1-8
1.2 Software Features	1-8
1.2.1 System Monitor/Debugger	1-9
1.2.2 I/O Subroutine Package	1-10
1.2.3 Line-Oriented Editor	1-11
1.2.4 Symbolic Assembler	1-11
1.2.5 Copy Utility	1-12
2.0 Tape Editor	2-1
2.1 Editor Files	2-1
2.2 Editor Commands	2-1

2.3 Editor Usage Notes 2-6
2.4 Editor Tape Error Recovery Procedures. 2-6

3.0 Tape Assembler 3-1

3.1 Running an Assembly 3-1
3.2 Assembly Options 3-1
3.3 Assembler Syntax 3-3

3.3.1 Label Field 3-4
3.3.2 Op-Code Field 3-4
3.3.3 Operand Field 3-5

3.4 Assembler Pseudo Instructions 3-8
3.5 Assembler Comments 3-11
3.6 Error Flags 3-11

4.0 Tape Debugger and Microemulator Debugger 4-1

4.1 Debugger Display Format 4-1
4.2 Debugger Commands. 4-3
4.3 Debugging Facilities 4-17

5.0 Operating the Microcomputer 5-1

5.1 The Tape Bootstrap Loader 5-1
5.2 Restarting the Monitor/Debugger 5-2
5.3 The "Reset" Button 5-2
5.4 Cassette Tapes 5-3
5.5 Cassette Tape Units 5-3
5.6 Error Recovery During Loading 5-3
5.7 CRT Adjustments 5-4
5.8 Machine Status Indicators 5-4

6.0 Debugging Facilities 6-1

6.1 Manual Interrupts. 6-1
6.2 Software Interrupts 6-1
6.3 Finding a Bug Using Software Breakpoints 6-2

7.0 Programming the Microsystem 7-1

7.1 Memory Configuration. 7-1
7.2 Memory Address Mapping. 7-1
7.3 Monitor/Debugger Interrupt Vector Usage 7-2

7.3.1 8080A and Z-80 Interrupt Vector Usage 7-2
7.3.2 6800 Interrupt Vector Usage 7-3

7.4 CPU Stack 7-3
7.5 Generating Bootstrap Loadable Tapes 7-4

8.0 Monitor I/O Service Routines 8-1

8.1 Keyboard Input 8-1
8.2 CRT Output 8-1
8.3 Cassette Input/Output 8-2

8.3.1 Cassette Write Routine 8-3
8.3.2 Cassette Read Routine 8-4
8.3.3 Cassette Read Retry Routine 8-6
8.3.4 Cassette Rewind Routine 8-6

8.4 EIA Input/Output 8-7

8.4.1 EIA Send Routine 8-7
8.4.2 EIA Receive Routine 8-8
8.4.3 Setup for ASCII Data 8-10

9.0 Direct I/O Programming 9-1

9.1 Keyboard Input 9-1
9.2 "Break" Key 9-2
9.3 CRT Display Select 9-3
9.4 CRT Display Output 9-4
9.5 Memory Protect 9-4
9.6 Real-Time Clock 9-5
9.7 EIA Communications 9-7
9.8 Parallel I/O Port 9-8
9.9 Summary of Interrupt Vectors 9-10

10.0 Appendix 10-1

10.1 Configuring Memory 10-1
10.2 System Program Debugger 10-3
10.3 Reassembling the Utility Program 10-3

1.0 Microsystem Overview

The Microsystem/10, 15, 20 and 30 are complete, low-cost microcomputer systems incorporating the 8080, 6800 or Z-80 single-chip CPU's. The basic system includes not only the processor and memory, but a complete set of peripherals and software carefully designed to make full use of the capabilities of the system.

Physically, the Microsystem/10 is composed of five units: mainframe, CRT display, keyboard, dual cassette tape unit, and optional line printer. The standard system comes with the CPU, 8K bytes of memory, a 960-character alphanumeric CRT display, a 53-key ASCII keyboard, and dual 2000 bps tape unit. Also included in the standard hardware is a bootstrap loader in PROM, memory write protect, two EIA RS-232 interfaces, an 8-bit parallel I/O port, and a crystal-controlled real-time clock.

Software for the systems includes a monitor/debugger, line editor, and an assembler, all designed to take full advantage of the high-speed multi-line CRT display and the tape I/O.

The Microsystem/15 provides all of the features of the Microsystem/10. In addition the Microsystem/15 is supplied with a total of 32 kilobytes of RAM memory and the Quickrun software package which provides space for Editor, Assembler, Monitor-Debugger and 18 kilobytes of user workspace. The Microsystem/15 will also support BASIC/8080 or BASIC/Z80.

While the Microsystems are ideal for developing microcomputer programs, they have many other applications as a complete, low-cost computer systems with high-speed display, mass auxiliary storage, and a software operating system.

1.1 Hardware Features

The Microsystems have a long list of standard features. This section describes the hardware features of the system to provide a framework for the sections which follow on programming and operating the microcomputer system.

1.1.1 CPU

1.1.1.1 8080 CPU

The Microsystems/10, 15, 20 and 30 Mod 8080 come with the 8080A CPU. This microprocesor has a repertoire of 78 instructions, including binary arithmetic, decimal arithmetic, logical operations, conditional and unconditional branches, and subroutine calls. The CPU contains seven 8-bit data registers, two memory

address registers, four testable flags, and an 8-bit arithmetic unit. The 8080A also contains a 16-bit stack pointer which controls the addressing of an external stack that can be used for saving machine status, register values, and subroutine return addresses.

The architecture and instructions of the 8080A CPU are fully described by the Intel publication "8080 Microcomputer System User's Manual."

1.1.1.2 6800 CPU

The Microsystems/10, 15, 20 and 30 Mod 6800 come with the 6800 CPU. This microprocessor has a repertoire of 72 instructions including: binary arithmetic, logical operations, conditional and unconditional branching and subroutine calls. The CPU contains 2 accumulators (8-bits), a 16 bit index register, and a 16 bit stack pointer which can be used for saving machine status, register contents, and subroutine return addresses.

The architecture and instructions of the 6800 CPU are fully described by the Motorola publication "M6800 Microprocessor Programming Manual".

1.1.1.3 Z-80 CPU

The Microsystems/10, 15, 20 and 30 Mod Z-80 come with the Z-80 CPU. This microprocessor has a repertoire of 158 instructions including all 78 of the 8080A microprocessor. In addition the Z-80 has instructions for searching and transfering blocks of data, instructions for bit manipulation, and expanded decimal arithmetic. The Z-80 has 17 internal registers including two 16 bit index registers.

The architecture of the Z-80 is described in the Zilog publication "Z80-CPU Technical Manual."

1.1.2 Processor Independent System Bus

The system bus of the four Microsystems has been designed around a clean, universal bus protocol so that CPU modules for the 8080A, 6800 and Z-80 can be accommodated without additional modification. Thus, memory and I/O devices will remain unchanged as CPU cards are interchanged. Both 8 and 16 bit microprocessors can be accommodated because 16 lines have been provided for the data bus. Memory modules contain 8K bytes of memory, but are strappable to function as the high-order or low-order 8 bits of a 16-bit memory. Thus two modules used together can be configured as 8K words of 16 bits.

1.1.3 Universal Bus Configuration

The bus is also designed so that bus signals are uniquely assigned to PC edge-connector pins. Thus all sockets on the mother board have the same signal assignments, and any PC module of the Microsystem will operate in any card slot position. There are no assigned card positions in the card cage.

1.1.4 Crystal-Controlled Processor Clock

For applications which require precise instruction timing, a crystal-controlled CPU clock has been provided. This clock operates at 2 MHz and provides a precise instruction subcycle timing of 500 ns for the 8080A.

The 6800 CPU is operated at 1 MHz. this provides a 1 us instruction subcycle. The Z-80 CPU is operated at 2.0 or 4.0 MHz. This provides an intstruction subcycle of 250 or 500 ns.

1.1.5 Bootstrap Loader in PROM

A bootstrap loader program resides in read-only memory (PROM). When the bootstrap program is invoked by the LOAD button in the keyboard, a program is loaded from cassette tape or floppy disk into memory. Special hardware associated with the bootstrap function allows the microprocessor to start up and execute the loader program which is located away from any privileged locations in memory. For the 8080A and Z-80 CPU's this allows read/write (RAM) memory to begin at location zero so that user programs can make use of the eight vector addresses required for using interrupts and the RST instruction.

For the 6800 CPU this allows page zero as well as the Reset, and Interrupt vectors to be in RAM and thus under user control.

1.1.6 System Reset

The system reset function, which is invoked by the RESET button, generates a system-wide reset signal that clears I/O device status registers and causes a master reset at the CPU. The 8080A or Z-80 then begins execution at location zero. Unless the proper instructions for restarting execution of the loaded program are stored beginning at location zero, the reset operation may have unpredictable results. The 6800 begins execution at the instruction whose address is stored in location X'FFFE'. A proper instruction sequence must exist at that address for predictable results.

1.1.7 Power-On Bootstrap

Following power-on of the Microsystem, a system reset is generated to clear status and then the bootstrap loader is automatically invoked.

1.1.8 Direct Memory Access

A direct memory access (DMA) channel is provided in the Microsystem which allows I/O controllers to directly access system memory. A maximum transfer rate of 1 megabyte/second is possible using DMA.

1.1.9 Priority Interrupts

1.1.9.1 8-Level Interrupts for the 8080A and Z-80

A priority 8-level interrupt system has been implemented. The interrupts are vectored, thus an interrupt causes the CPU to branch to one of eight fixed locations in low memory. The interrupt vector addresses correspond to the eight locations reached using the eight RST instructions. Since the interrupts are priority encoded, if more than one interrupt request is made simultaneously, the processor will respond to the interrupt having the lowest vector address; i.e., highest priority.

1.1.9.2 4-Level Interrupt for the 6800

The 6800 has a 4-level interrupt system. A series of four address vectors are stored in the eight highest locations in memory (X'FFF8' to X'FFFF'). These implement the maskable interrupt, software interrupt, non-maskable interrupt and system reset.

1.1.10 System Interrupt Mask

The 8080A CPU has an interrupt enable flip-flop which allows processing of interrupts when it is set and inhibits interrupts

when it is reset. It may be directly set and reset by the enable (EI) and disable (DI) interrupt instructions. However, whenever an interrupt is processed by the CPU, the enable flip-flop is automatically reset. This allows the interrupt to be processed without the CPU being interrupted again. The enable flip-flop is also reset by the master system reset.

The 6800 and Z-80 CPU's have an interrupt enable flip-flop which allows the maskable interrupt request to be enabled or disabled but does not effect the non-maskable interrupt or software interrupt.

1.1.11 I/O Device Interrupt Masks

Each of the I/O devices of the Microsystem has a separate interrupt mask flip-flop. This allows the software to selectively enable and disable I/O device interrupts. The master system reset clears mask flip-flops. Thus all devices have their interrupt masks disabled initially.

1.1.12 8K Memory Modules

The memory of the Microsystem is made up of 8K byte modules of read/write (RAM) memory. These modules are implemented using 4K RAM chips. The basic system contains one 8K module. The memory is expandable by plugging in additional modules.

1.1.13 Memory Write-Protect

Each 8K memory module has a write-protect register which allows for selective write-protection of each 1K page of memory. When a page is write-protected, the data can be read from the memory, but new data cannot be stored into that memory page. This feature is useful in protecting system or user programs from being clobbered by programs under test.

The protection flags for the 8K module are stored in an 8-bit write-protect register which may be read or reloaded under software control. All protection flags are cleared by the master system reset.

1.1.14 CRT Display

The Microsystem includes a CRT controller module which produces a 960-character alphanumeric display on a standard low-cost television monitor. The CRT module makes use of direct memory access (DMA) to obtain the data to be displayed from the system memory. The characters are displayed on the CRT as 24 lines of 40 characters. The display is refreshed at a rate of 60 times per second. Therefore, data stored in the display memory will appear

on the CRT within 1/60th of a second. The data rate to the CRT is limited only by the rate at which the processor can store data in the memory (approximately 20,000 characters/second).

1.1.15 CRT Display Memory Page Select

The 1K memory page from which the CRT display is refreshed can be selected (under software control) as any 1K page in the Microsystem memory. This allows the software to select the area of memory to be used for the CRT display and allows rapid switching between several displays already formatted in memory.

1.1.16 53-Key ASCII Keyboard

The Microsystem is supplied with a 53-key ASCII encoded keyboard. It is a reed-switch unit with 2-key rollover. Each keystroke generates a strobe which can be read as a status bit by the CPU. When enabled, the keyboard can also generate an interrupt for each key depression. This feature is inhibited by its mask bit when it is not required.

The keyboard also has a programmable function key which does not generate an interrupt, but can be read directly by a program at any time. This key is labelled "HERE IS,".

1.1.17 Breakpoint Interrupt

The keyboard provides a BRK key which generates a higher priority interrupt than the keyboard and is enabled or inhibited by a separate mask bit. The BRK key is used to generate a breakpoint interrupt, which is used by the debugging software so that execution of a program can be manually interrupted and then restarted. The breakpoint interrupt can also be used in application programs as an escape key.

1.1.18 Load, Reset Buttons and Status Display

A mini-front panel has been implemented on the Microsystem keyboard. This mini-console has two switches and two status display LEDs. The LOAD button invokes the PROM resident bootstrap loader, while the RESET button generates a master system reset. The two LED displays indicate "run/halt" and "interrupts enabled/disabled for the 8080A." When lit, they indicate "run" and "interrupts enabled" respectively.

The "interrupts enabled" display is used as feedback during cassette tape load and dump for the 8080A, 6800 and Z-80 CPU's. For the 6800 or Z-80 it does not indicate the status of the CPU interrupt enable flip-flop.

1.1.19 Cassette Tape Units

The standard Microsystem/10 and 15 come with a dual cassette tape unit. These units are low-cost audio cassette units and record on audio quality Phillips cassette tapes. The FUTUREDATA interface does not require any modification to the tape units for digital recording. Data is recorded at a rate of 2000 bits/second. While the inteface provides for the on/off control of the tape unit motor, the "rewind," "read," and "write" modes are manually selected by the operator. The cassettes can be individually write-protected by a knock-out plastic tab.

Error detection, error recovery, and record formatting are under software control. A tape I/O software driver in the system monitor/debugger provides for a high-level software interface to the tape units. The soft error rate observed from the tape units is approximately one bit in 1xE8. The hard error rate, after the error detection/correction has been applied, is approximately one bit in 1xE14.

The data rate for loading binary data into memory after error detection overhead has been taken into account is 200 bytes/second or 1600 bits/second. Loading of a 4K byte segment takes only 30 seconds including the 10-second leader.

1.1.20 EIA Interfaces

All Microsystems have, in addition, two EIA RS-232 compatible interfaces. One interface, designated "EIA 1," is configured for communicating with a terminal. While the transmitted data and received data lines may be controlled by the microcomputer, control lines to the terminal are permanently enabled, and control lines from the terminal are ignored. The data interfaces to the microcomputer are at the bit level. An EIA I/O software driver, in the system monitor, provides a high-level software interface to the EIA lines. Serialization, deserialization, and parity detection/generation are all provided by the software package. Baud rates from 50 to 1800 are program selectable.

The second interface, designated "EIA 2," is configured for communicating with a modem. In addition to the transmitted data and received lines, the microcomputer controls the request-to-send and clear-to-send lines. If the received data line is not used on "EIA 1," it may be strapped to connect to the carrier-detect line of "EIA 2."

1.1.21 Real-Time Clock

In order to precisely time serial data on the EIA lines as well as for other applications, a real-time clock has been implemented in

the Microsystem. The real-time clock is derived from the crystal-controlled CPU clock and has six bits of resolution. The least significant bit represents 32 microseconds, and the clock overflows every 2.048 milliseconds. Whenever the clock overflows, this sets a status flip-flop which can, if enabled, interrupt the microcomputer. Again, this interrupt is maskable and is disabled by the master system reset.

The real-time clock overflow period can also be set under software control. In setting the clock, eight bits are used, giving the least significant bit 8 microseconds resolution.

1.1.22 8-Bit I/O Port

A universal 8-bit I/O port is also provided on the Microsystem which has general application in connecting the system to external equipment and devices. The port has eight bidirectional TTL data lines. Output data can be latched in the port and two output strobe lines are provided. Two input strobes are also provided which allow the processor to read data from the port. In addition, a status latch holds an incoming acknowledgement strobe for interrogation by the Microsystem CPU. This status line can interrupt the processor if enabled.

1.2 Software Features

To complement the long list of standard hardware features, the Microsystem software is equally comprehensive. The software systems make full use of the high-speed CRT and tape or disk units to enhance the usefulness of the microcomputer system.

The Microsystem/10 software package is composed of four major elements: a system monitor-debugger, a line-oriented editor, and a symbolic assembler and a copy utility.

The Microsystem/15 provides the same set of systems software in a single integrated package called QUICKRUN. QUICKRUN is a proprietary FUTUREDATA software package which is used to speed the development of software for the 8080, 6800 and Z-80 microprocessors. QUICKRUN requires 32K bytes of memory for operation, and provides the three basic software development programs: Editor, Assembler and Monitor/Debugger all in memory at the same time. The Z-80 and 8080 versions of QUICKRUN contain the full MICROEMULATOR debugger.

The QUICKRUN system contains the same Editor, Assembler and Monitor/Debugger programs that are available for the basic tape based FUTUREDATA system. QUICKRUN greatly enhances the basic system by eliminating most of the tape operations involved in using the tape based system. The QUICKRUN system tape is loaded

once at the beginning of a work session and brings in all three software programs at once.

When loaded, QUICKRUN occupies 14K to 16K bytes of the 32K byte memory. This leaves 16K to 18K of user memory for the editor workspace, the assembler symbol table and user object code. More memory can be added to a maximum of 48K to 64K. This allows the development of programs as large as 2500 lines without any tape input and output.

Because all of the software is in memory at once, control can be transferred from Editor, to Assembler, to Monitor/Debugger all instantaneously. A development session is begun by loading source text of a program into the editor work area. Once the source data is loaded, it is unnecessary that it be again reloaded for the duration of the session.

After a program is loaded by the editor, the assembler can then be used to convert the source code into binary, executable data. While all of the tape operations are available, the assembler is used most conveniently to take the source program from memory (in the editor work area) and place the object program in memory. This is a fully memory based operation, no tapes need be involved. The assembler will process a 1000 line program in about 20 seconds in the memory based mode.

Once assembled, debugging can begin using the Monitor/Debugger. The program has already been assembled directly into memory. The memory write protection feature is utilized during debugging to prevent the test program from changing data in the system area, which includes the editor work area.

If program errors are found, it is exceptionally easy to just use the editor to make source code changes and reassemble in seconds.

In order to provide a framework for later sections, the major functions of the software systems will be summarized here.

1.2.1 System Monitor/Debugger

The Microsystem/10 and 15 Monitor/Debugger is composed of three software subsystems: a program loader, an interactive debugger, and an I/O subroutine package. The program loader is more comprehensive than the abbreviated bootstrap loader in PROM, and performs the task of loading binary, executable programs from cassette tape into the system memory. The loader includes a dump routine which performs the function of writing a binary memory image out onto cassette tape for permanent storage. The dump routine can perform a scatter dump, which means it can write a number of noncontiguous sections of memory out onto tape. The

loader routine, when processing such a tape, restores the data to the sections of memory from which it was dumped.

The interactive debugger, provided by the system Monitor, allows the programmer to display data in the microcomputer memory and store changes or corrections. Data in the memory is displayed by the debugger in hexadecimal and, when applicable, in ASCII form. Changes to be stored in the memory are entered in hexadecimal form. The debugger also allows for initiating the execution of a program in memory. If the program does not function correctly, execution can be interrupted by the BREAK key, and the CPU status is saved and displayed by the debugger. If desired, the machine status can be restored and execution can resume by a command to the debugger.

A breakpoint instruction can be stored at a critical point, in a program under test, using the debugger. When execution reaches the breakpoint instruction, control is returned to the debugger and the machine status is saved and displayed for evaluation.

1.2.2 I/O Subroutine Package

The system Monitor/Debugger also contains a series of I/O driver subroutines which provides high-level interfaces to the various standard FUTUREDATA I/O devices.

Included in the subroutine package are cassette tape read/write routines. These provide a record-oriented interface to the tape units.

A keyboard input routine and a CRT output routine allow easy use of these devices. The CRT output routine performs the functions of a Teletype-compatible CRT terminal. Characters are passed to the routine one at a time, and the control codes "carriage-return," "line-feed," and "backspace" are properly handled. While the CRT is capable of direct addressing in a much more versatile way, the CRT output routine provides a convenient interface to the CRT display which is adequate for many applications.

EIA line input/output subroutines are also a part of the I/O routine package. These routines perform serialization and deserialization of raw EIA data on a character-by-character basis. The routines transmit and receive data at all standard rates between 50 and 1800 Baud. The routines also optionally generate and detect odd or even parity.

The present revisions of monitor/debugger do not contain the CRT, or EIA subroutines. This is done to conserve space in the basic 8K system. These routines are provided on the "I/O Subroutine

Source" tape for inclusion in user software.

1.2.3 Line-Oriented Editor

The Microsystem/10 and 15 Editor provides a means for preparing and updating symbolic data stored on cassette tapes. Under control of the microcomputer programmer, data to be edited is read from an input tape into a memory work area. When data is entered for the first time, it may be entered into the memory directly from the keyboard.

Once data is in the memory, it can be displayed and edited. Editing commands allow the display to be scrolled through the lines in memory, or a search can be initiated for a specific character string. Lines can be inserted, deleted, or modified. Character-editing commands allow character insertion, deletion, or replacement within a single line.

Once editing of the data in the memory work area is complete, it is written out onto an output tape. Since the memory work area is not always large enough to hold an entire data file at one time, the process described above (of reading, editing, and writing) can be repeated as many times as necessary to work through the whole data file.

1.2.4 Symbolic Assembler

The Microsystem Assembler runs on the microcomputer system and converts source programs written in symbolic assembly language into binary, executable form. Programs written in assembly language make use of mnemonic names for the machine instructions and assembler psuedo-instructions. The assembler translates the mnemonic instructions into binary machine codes. Symbolic labels are also used to represent addresses in the source program; the assembler converts the symbolic addresses into absolute binary memory locations.

The assembler processes the source program in two passes through the input. The source program is provided on cassette tape and is generated by the editor. During pass 1, the assembler reads the program and builds a symbol table which equates absolute binary values and memory addresses with the symbolic labels used in the source program. During pass 2, the assembler uses the symbol table, and while reading the input tape a second time, translates each source program statement into binary object data. The Microsystem/15 allows the two passes to be performed on source data in memory. This greatly speeds the assembly operation.

A number of options are available during pass 2 of the assembly. A listing of the program can be requested for display on the CRT.

Any errors detected by the assembler are flagged in this listing. A hardcopy of the program listing can also be requested, and a loadable binary object tape can be written. The object tape can later be loaded by the system monitor/debugger which brings an executable, binary translation of the symbolic program into memory ready for execution or debugging.

1.2.5 Copy Utility

The Microsystem Tape Utility provides the capability of transferring (or copying) data between the various media supported by the Microsystem/10 and Microsystem/15.

In the first release of the Utility, only the transfer of data between cassette tapes is implemented. New releases of the utility will be made available as additional functions are added. However, the complete source code for the Utility is available, at nominal charge, so that the user can implement drivers for addtional or custom I/O devices.

When the utility is loaded, a menu of functions is displayed. Source file copying is selected by "S" and object file copying is selected by "O".

Once the selection of the source or object copying procedure is made, the utility requests "SPECIFY INPUT DEVICE". At this point a "T" indicates tape input (from unit 1). Other devices will be recognized in the future.

Following specification of input, the utility program requests "SPECIFY OUTPUT DEVICE". Again, a "T" specifies output to tape (unit 2).

When all of the parameters have been specified, the data transfer commences. The cassette units should be readied before the parameters are entered, since the copy procedure begins as soon as the last parameter is entered. When finished, the message "FUNCTION TERMINATED" is displayed and the message "REWIND" also appears. The "RETURN" key is used to acknowledge that rewinding is complete.

At any time during the data transfer, the operation can be interrupted by pressing the "RETURN" key. Response may not be immediate, but as soon as the function comes to a clean stopping point, the transfer will be terminated. The message "FUNCTION ABORTED" will be displayed along with the "REWIND" message.

If an unrecoverable error condition is dectected during the data transfer the message "FUNCTION ABORTED" will appear. The message "ERROR. . RETRY" will be displayed. After pressing the RETURN key,

rewind the input tape for a couple of seconds and then return to play mode so that the data can be reread. When the error message is displayed, if an "E" is typed instead of a RETURN, the function will be terminated.

If an error is detected while reading an object tape, the operation must be retried from the beginning, and the message "ERROR.. REWIND" appears.

2.0 Tape Editor

The FUTUREDATA tape Editor is bootstrap loaded from the input cassette tape. Once loaded the Editor may be used to edit a series of source tapes.

The FUTUREDATA QUICKRUN operating system editor is entered from any of the system functions using the "JE" command. The editor provides the capability for creating and updating source (text) files on tape and in memory.

The Editor display shows 21 lines of text. The line presently being edited or entered is displayed in the center of the screen between an upper and lower dividing line. The present tab stops appear as "+"'s on the top dividing line. The present position of the edit cursor appears on the bottom dividing line. Commands are entered at the bottom of the screen at the command cursor "> ". Some commands ("X" and "I") will remove the command cursor and display an edit cursor. At this point data is being entered or changed in Editor work area. When the command cursor appears at bottom of the screen the Editor is ready to accept commands as differentiated from data.

2.1 Editor Files

Editing data on tape involves two files, an input tape file and an output tape file. Data is read from the input file, edited in the memory work area, and written into the output file.

Whenever editing is performed, the current data is read (loaded) from the input tape file and the updated data is written into the output tape file.

2.2 Editor Commands

The FUTUREDATA editor is a CRT based editor. All data is shown in context with the line presently being edited ("editing line"), in the center of the screen. Ten lines preceeding the editing line and ten lines following the editing line are always shown on the screen. A command entry line at the bottom of the screen is used to enter the following commands.

The following is a list of the Tape Editor commands.

COMMAND	FUNCTION
L	Specifies that the input is to come from the

input tape file and begins loading the editor work area with data from the file. Reading will stop when the work area is full, or when the end-of-file is reached. After writing all or a portion of the editor work area subsequent "L" commands will read additional data from the input tape file. If the file has been read completely through, the message "END-FILE" will appear.

W — Specifies that all data from the editor work area is to be written into the output tape file. When the command completes, the work area is cleared and is ready for additional data. Subsequent entries of the "W" command initiate the transfer of additional data to the output tape file.

W$ — Specifies a write of only the data preceding the current location of the editing line.

WB — Output to both tape files. This is useful in the first entry of the data to create two copies of the source program.

WB$ — Output the top part of the editor work area to both tape files.

N — Performs a "W" followed by an "L".

N$ — Performs a "W$" followed by an "L".

E — Indicates that this is the last write command to the output tape file. It will cause the contents of the editor work area to be written to the output tape file. An "end of file" record will be appended to the file and the message "REWIND OUTPUT" will appear. The user should rewind the output tape and indicate that the operation is complete by typing a "CARRIAGE RETURN". Note: THIS MUST BE DONE PRIOR TO REMOVING THE OUTPUT TAPE OR THE EDITOR POINTERS WILL NOT BE CORRECT AND SUBSEQUENT EDITING SESSIONS MAY PRODUCE WRONG RESULTS.

EC — This command performs a check write and is especially useful for the QUICKRUN Tape system. The "EC" command will write the data and an End-file to the output tape file. It will also request a rewind and reset the Editor pointers. The source program is left in the editor work area and can be assembled directly from memory.

	This is an exeeedingly fast operation and is the perfered mode of use in the QUICKRUN system, whenever the source program will fit into the available memory.
EB	Output to both tape files and append an End-file. This command will request a rewind and reset the Editor pointers.
REW	Rewind the input tape file. Note: THIS MUST BE DONE PRIOR TO REMOVING THE CASSETTE FROM THE INPUT TAPE DRIVE. The system will request the user to rewind the input tape file by displaying "REWIND INPUT". The user then rewinds the input tape file and acknowledges the system request by typing a "RETURN". If the message "END-FILE" appears in response to an "L" or "N" command, then the "REW" must be given prior to any further attempts to load data.
A	Advance one line.
A n	Advance "n" lines. The parameter "n" must be less than 256.
A $	Advance to the end of the workspace.
CTRL A	Advances one line. This does not require a following return as the usual "A" command does.
CTRL A REPT	Advance or scroll up at the rate of about 10 lines per second.
B	Back up one line.
B n	Back up "n" lines. The parameter "n" must be less than 256.
B $	Back up to the beginning of the workspace.
CTRL B	Back-up one line. This command does not require a following return.
CTRL B REPT	Back-up or scroll down the display at the rate of about 10 lines per second.
DEL	Delete the line displayed as the editing line.
DEL n	Delete the line displayed as the editing line and the next "n"-1 lines. the parameter "n" must be

less than 256.

X Change the data in the editing line. This command provides a number of line-editing commands which allow the editing line to be modified. These editing subcommands control the modes of interaction which may catagorized as: (1) non-destructive cursor positioning, (2) character replacement and (3) character insertion and deletion.

When the "change" (X) command is invoked, the editor enters the non-destructive cursor positioning mode. Space, backspace and tab keys may be used to position the non-destructive cursor " ". A summary of the key functions is:

space

Moves the cursor ahead one character. The characters in the edit line are unaffected.

backspace

Moves the cursor back one character. The characters in the edit line are unaffected.

tab

Moves the cursor ahead to the next tab stop.

R

Enter the replace mode. All subsequent characters are destructive except line-feed. Line-feed will return to the non-destructive cursor positioning mode.

I

Enter the character insert mode. A line-feed will return to the non-destructive cursor positioning mode.

D

Delete the character above the cursor.

CTRL X

Cancel the entire line.

carriage return

Exit from the line editing command.

I

Insert data into the work area ahead of the editing line. When the insert command is executed, the editing line and lower half of the screen are rolled down, the editing line is blanked and a cursor ("+") appears at the left of the editing line. As each line is typed and a carriage return is entered (to signal the end of the line), the editing line and top half of the screen are rolled up one line. The editing line is again blanked and the next line can be entered. If a carriage return is is entered prior to any other character on the line, then the insert mode is terminated.

All of the line editing subcommands, described above, are available in the insert mode.

F /string/

Find the character string "string" in the editor workspace. The search begins at the line following the edit line. If a match is made, the line containing the string will be positioned in the editing line.

If no match is found the message "NOT FOUND" will be displayed and the editing line will be positioned to the end of the edit workspace.

F

This form of the find command uses the same string as last entered.

G /string/

The resume command is a continous search command that will write out the present contents of the workspace search through subsequent loads from the input tape file.

G

This form of the resume command uses the same string as previously entered.

CLR

Clear all data from the editor workspace.

T #,#,...,#

Set the tab stops. The tabs are set at the columns entered as parameters. The tab stops may be set between 2 and 40. The default values of the tab stops are at 8, 13, and 23. The tab

stops are indicated by the "+" signs displayed on the top divider of the editing line.

J function — Transfer control to other system functions (QUICKRUN only).

2.3 Editor Usage

To provide the ability to recover from operator errors after reading or writing has begun, a means for interrupting these operations has been included.

During a reading ("L") or load operation, processing can be interrupted by pressing the RETURN key. Reading does not stop instantly, but instead continues until the end of the data record being read. Reading can be restarted if necessary using the "L" command.

This method of interrupting a read operation, which provides for the ability to restart, will not work in the case when an end-of-file is missing from a tape, and the editor is trying to read data beyond the logical end-of-file. In this case, the BRK can be used to abort the read operation. If the BRK key is used instead of the RETURN key while valid data is being read, some data in the record being read will be lost.

During a write ("W" or "E") operation, processing can also be interrupted by pressing the RETURN key. When the editor finishes with the record being written, it will display the message, "RESTART WRITE?" If the desire is only to interrupt the write operation, then a RETURN at this time causes the editor to delete just the data already written to tape, and return to command entry mode. This allows the write to be restarted later without any duplication of data. However, if a "yes" answer is given (Y key is pressed) then the editor will rewind the output tape and stop the write without deleting any data from the editor work area.

Interruption for rewind and restart (i.e. "Y" answer to RESTART WRITE message) can only be used during the first write to an output tape. This form of write interruption is only useful to recover from the situation where a write onto a new tape is begun and the tape jams or is not set up properly. Once data has been written onto the output tape, the "RESTART" must not be used or data will be lost.

2.4 Editor Tape Error Recovery Procedures

Some situations exist -- mostly due to operator errors -- which cause the Editor to enter an error state. Most errors do not cause any loss of data, and if data is lost, it is unlikely to be

more than 10 lines.

When an error is encountered during a read operation, it is indicated in one of three possible ways:

1. ERROR. . . NO DATA OR WRONG TAPE message,
2. "ERROR. . . RETRY" message,
3. Read never completes.

Procedure when ERROR. . . NO DATA OR WRONG TAPE is displayed:

This message means that the input tape is blank or does not contain editor-compatible records. If the end-file is missing, this message may occur when the editor attempts to read past the end of the recorded data. In either case, enter a RETURN to return control for further editing.

Procedure when the message ERROR. . . RETRY is displayed:

In the unlikely event that an "ERROR . . . RETRY" message appears, and is not due to an operational error like a wrong tape, the retry sequence is as follows. The input tape will stop when the message is displayed. Place the input unit in rewind mode, then press the RETURN key. The recorder will be energized for rewind. In only a few seconds, press the STOP key and restore the recorder to PLAY mode. This sequence will back up the tape, so that the read can be attempted again. If the error condition remains then entering an "N" will cause the editor to skip to the next readable record on the tape with some loss of data.

Procedure when reading never completes:

When reading does not complete, due to a variety of operator errors, the BREAK key will return control for further editing.

One very important rule will avoid many problems. It is as follows:

NEVER REWIND, REMOVE OR REPOSITION AN INPUT OR AN OUTPUT TAPE WITHOUT USING THE REWIND ("REW") AND END-FILE ("E", "EC" OR "EB") RESPECTIVELY. Because the Editor needs to know when new input and output tape files are loaded, status infomation is derived from the use of these

commands. Violation of this rule may invalidate the Editor's tape-status information, leading to loss of data.

3.0 Tape Assembler

The FUTUREDATA Tape Operating System Assembler is capable of reading assembly language source files and performing the translation into machine executable object files.

3.1 Running an Assembly

Upon entry to the assembler using a "JA" command, for the QUICKRUN system or after loading the assembler tape for the stand alone tape systems, the assembler awaits input of a series of option selections.

The QUICKRUN Assembler automatically runs pass 1 and then pass 2. The tape assembler will generate the message "REWIND INPUT" after pass 1 and await a user RETURN to begin pass 2. The user should rewind the tape, type RETURN and put the input tape drive into read mode.

During the assembly, pressing the RETURN key will cause the assembler to pause and display the message "RESTART?" If the assembly is to be aborted, a "Y" (Yes) answer will terminate the assembly. A RETURN will allow the assembly to continue.

3.2 Assembly Options

The assembler recognizes the "J" command (QUICKRUN ONLY) when it is requesting assembly options. At any point in the option specification process, the "CTRL X" key can be used to erase the current options and provide for respecification or exiting from the assembler.

The list of assemly options is as follows:

OPTION	FUNCTION
J function	Transfer control to another system function. (QUICKRUN only)
L	Produce a program listing on the CRT.
T	Produce a program listing on the CRT with lines truncated to 40 characters. When hardcopy is requested also, the "T" specifies that only the first line of a "DC" instruction is to be printed, but does not truncate the length of the lines.

E — Only lines containing errors flagged by the assembler are displayed on the CRT if "L" or "T" is specified, or printed if "P", "B" or "K" is specified.

O — Specifies that an object data tape file is to be written.

I — Specifies to the QUICKRUN assembler that input is to come from the input tape file.

N — Specifies to the QUICKRUN assembler that the default option of storing the object data into memory is to be inhibited. This is used to remove the "A" error flags when an attempt is made to load into protected memory.

Z — This option will flag all lines with non-8080 or 8085 opcodes. These lines will be flagged with a "P". This applies to the Z-80 assembler only.

P — Print a program listing on the line printer. (Centronics with form-feed option). "T" will not affect the printed copy, but "E" will cause only error lines to be printed.

P1 — Print on Centronics printer without form-feed option.

K — Print listing on Okidata CP-110 printer with RS-232C option.

B — Selects hardcopy output to the EIA port 1 for RS-232C compatible printing terminals. The transmission rate can be selected by a subsequent digit as shown in the table below. If the rate selection digit is not given, the transmission rate is the same as used in an immediately previous assembly. When the Assembler is first loaded, the default rate, if none is specified, is 110 Baud.

0 - 50 Baud
1 - 75 Baud
2 - 110 Baud
3 - 134.5 Baud
4 - 150 Baud
5 - 300 Baud
6 - 600 Baud
7 - 900 Baud
8 - 1200 Baud
9 - 1800 Baud

When transmission rates less than 300 are specified,

only 1 null is sent after a carriage-return line-feed sequence. For transmission rates 300 or greater, 4 nulls are sent.

If a second parameter, after B (0 to 9) is specified then the number of null characters will be altered accordingly.

If the assembler encounters an End-of-file without an END statement, it will print the message "NEXT TAPE..."E" TO END". If the user types "E" the assembler will automatically insert an END statement. Otherwise, the user may insert another tape into the input tape drive and continue the assembly.

3.3 Assembler Syntax

Assembly programs are composed of three types of statements: (1) executable statements, (2) pseudo statements or instructions, and (3) comment statements. Each statement is one line of assembly code.

Assembly language statements are made up of four fields: (1) label field, (2) operation code field, (3) operand field, and (4) optional comment field.

The label field optionally contains a symbolic name which may be used when referencing the statement from other statements in a program.

The operation code or opcode field contains an 8080, Z-80, 6800 or 8085 mnemonic name or an assembler pseudo-instruction name.

The operand field contains additional data (e.g., immediate data or address) required for the specified opcode. Some instructions do not require any operand data.

The comment field is the last field and allows for annotation of the program. This field is not processed by the assembler, but is printed in the program listing.

The assembler statements have free fields, which means that the fields of the statement do not have rigidly defined lengths or positions. Instead, each field is separated from the next by one or more blanks. However, it must be noted that if no label is provided on a statement, at least one blank must still precede the opcode field.

The fields of the assembler statement will be represented diagrammatically in later discussions as follows:

Label Opcode Operand Comments

The maximum length of an assembler statement is 79 characters. Paragraph 3.5 describes the method for extending assembler statements beyond 40 characters.

The user should refer to the appropriate Assembler Reference Manual for details on the Assembler for the specific processor.

3.3.1 Label Field

A symbolic label may be supplied on statements which must be referenced from other statements of a program. The symbolic label is optional and need not be coded in every statement. When the label field is omitted, as is usually the case, at least one blank must precede the opcode field.

Valid symbolic labels are 1 to 8 characters long. The first character of the label must always be alphabetic, but other characters may be alphabetic or numeric. The delimiter between the label field and the opcode field can be a blank " ". Examples of valid labels are:

LOC1
LOOP
MXLENGTH
A123

During pass 1, the assembler assigns values to symbolic labels. The symbols and their assigned values are stored in a symbol table in the microcomputer memory. Since the symbol table is designed so that shorter symbols use less space and longer symbols use more space, it is advised that short symbol names be used where possible. This will conserve space in the symbol table and will allow a greater number of symbols to be accommodated.

Normally the value assigned to a symbol is the absolute memory location of the instruction statement on which the label occurs. However, a special assembler pseudo-instruction, EQU, described in detail in Section 3.4, allows for the symbol appearing on the statement to be assigned the value of the operand field.

It is recommended that the user avoid symbols that may conflict with valid microprocessor operands. These include register designators such as: A, B, BC, SP, X, or AF; and condition codes such as Z, NZ, P, or M.

3.3.2 Opcode Field

The opcode field contains the name of a microprocessor machine

instruction to be generated by the statement, or an assembler pseudo-instruction. Each of the machine instructions of the microprocessor has been assigned a 3 to 4-character mnemonic name. The instruction set mnemonics used by the FUTUREDATA Assemblers are the same mnemonics assigned by Intel, Zilog or Motorola. One of the main functions of the assembler is to convert the easy-to-remember symbolic instruction names into the binary machine codes.

Also appearing in the opcode field are assembler pseudo-instructions. These are mnemonic names which do not generate executable instructions, but instead command the assembler to perform special functions. The assembler pseudo-instructions will be detailed in Section 3.4.

3.3.3 Operand Field

The operand field is used to supply parameters (e.g., immediate data or addresses) which are required for the assembly of some instructions. The operand field is separated from the opcode by one or more blanks. Depending upon the opcode, the operand field may be omitted or may contain one or more data elements. The data elements within the operand field are separated by commas only. Since a blank is used to separate the operand field from the comment field, a blank inadvertently included in the operand field will cause the data following the blank to be ignored as a comment.

There are six ways of specifying data as follows:

Decimal constant
Hexadecimal constant
Current location counter
ASCII character
Value of symbolic names
Expressions

All operands are computed as 16-bit values. Only the number of bits implied by the data type are used in assembling the instruction. The operand value is taken from the low-order part of the computed value and high-order data is ignored.

The various operand types are:

TYPE	DESCRIPTION
Hexadecimal Constants	Hexadecimal constants are made up of the

hexadecimal digits 0-9 and A-F and must be prefixed by an "X" and enclosed in single quotes.

Single-byte values may be one or two digits long while two-byte (address) values may be one to four digits in length. When less than the maximum number of digits are used, the numbers are treated as if leading zeros were supplied; i.e., they are right justified.

Examples of single-byte hexadecimal constants are:

```
X'1'  (= X '01')
X'A3'
```

Examples of two-byte (address) constants are:

```
X'103'  (= X '0103')
X'BF2A'
```

Decimal Constants

Decimal constants are simply numbers made up of the digits 0-9. Single-byte decimal values may range from 0 to 255. Two-byte decimal values may range from 0 to 65,535.

Current Location Counter

The symbol "*" is used to represent the current value of the location counter. This is the address of the first byte of the assembled statement in which the "*" occurs.

ASCII Characters

Character constants store the code for symbolic characters as defined by the American Standard Code for Information Interchange (ASCII). An ASCII character constant must be enclosed in single quotes.

An ASCII character occupies a single byte. The ASCII code is a 7-bit code; the eighth (high-order) bit is zero when stored in an 8-bit byte.

Examples of ASCII character constants are:

'A'
'*'
''''

Symbol Values

Values assigned to symbolic names may be used as operands in addition to the constants described in previous sections. During pass 1 of the assembly, each symbol occurring in the label field is assigned a numeric value. The value assigned to a symbol is either the address of the statement at which it occurs or, for the EQU instructions (see Section 3.5), the value of the operand is assigned to the symbol.

Values of symbols are stored in the symbol table during pass 1. During pass 2, whenever the symbol is used in an operand, the assigned value is retrieved from the symbol table.

Expressions

Operands may also be defined as expressions composed of the constants or symbols.

Two binary operators, plus and minus, allow for the combinations of values. Multiple operators are allowed, and such expressions are evaluated from left to right. Parentheses may not be used for grouping elements for evaluation. Examples of expressions are:

```
ALPHA+25
*-3
X'36'-1
ALPHA-X'2E'+1
```

The value of the expression

```
2+6-3
```

is 5, as a result of the left-to-right evaluation.

Expressions are always evaluated as 16-bit quantities, though fewer bits may be required for some operands.

Several unary operators are also provided. They are:

```
H(expression)
L(expression)
```

The H (high-byte) and L (low-byte) operators are used primarily for single-byte operands. The "H" operator extracts the high-order byte of the constant, symbol, or expression in parenthesis. Smilarly, the "L" operator extracts the low-order byte of the constant, symbol, or expression in parenthesis. For example:

```
H(ALPHA)
L(X'2E35'+2)
```

3.4 Assembler Pseudo-Instructions

There are nine instruction mnemonics which are not machine instructions, but which are treated as commands to the assembler. These assembler commands perform extremely useful functions. They are:

INSTRUCTION	FUNCTION
END	The END statement is used to signal the end of assembler input statements. This statement must always be supplied as the last statement of a program. The operand field of the END statement should contain an address operand. This address specifies the program entry point. That is, when the program is loaded by the monitor, the entry pointer will be set to this address. The execute command can then be used to begin execution at the address specified as the entry point.
SPC	The space instruction is used to increase the readability of the assembler listing. This statement is replaced by a blank line in the listing. No operands are processed.
EJE	The eject instruction is also used to format the program listing. Eject causes the assembler to skip

to the top of the next page. No operands are processed.

EQU

The equate instruction is used to assign a value to a symbolic name. The symbol in the label field is assigned the value of the operand field, evaluated during pass 1. Because the value is assigned to the symbol during pass 1, it may be used like any other symbol and may be referenced in the operand fields of statements occurring before and after the equate in which it is defined. However, there is a restriction on the symbols that may be used in the operand field. Symbols referenced by the EQU instruction must have been defined (i.e., occur in the label field) in some statement prior to the equate statement.

Examples:

```
DATA        EQU         X'3F'
DATA1       EQU         DATA+26
```

ORG

The origin pseudo-instruction provides the capability for specifying the address in memory at which the subsequent statements are assembled. Normally sequential statements are placed in sequential memory locations during the assembly process. The origin instruction alters this by specifying that the subsequent instruction should be assembled at the address specified in the operand field.

The operand for the ORG instruction may be a constant, symbol, or expression representing a 16-bit machine address. As with EQU, symbols used in the operand field must be defined in a previous statement.

If no ORG instruction is given as the first statement of a program, the assembler begins with location 0.

Examples:

```
            ORG         X'1000'
            ORG         *+10
```

DC

The DC pseudo-instruction is used to define constant data to be placed in memory. Both single-byte and two-byte (address) elements may be generated by the DC instruction.

The operand field of the DC instruction may contain

one or more valid operands; i.e., constants, symbols, or expressions. When a list of operands is specified, each operand must be separated from the next by a comma. A blank delimits the end of the list, consequently no blanks are allowed within or between operands, except inside an ASCII string constant. The operands within a list are stored in sequential memory locations.

While operands for other statements only allow a single-byte ASCII character constant, the DC instruction allows ASCII character strings of varying length. ASCII strings are enclosed in single quotes. A single quote within the string is represented by two single quotes occurring sequentially. Each ASCII character occupies a single byte of memory. The characters within the string are stored in sequential bytes of memory.

Operands of the DC instruction are normally each stored in a single byte of memory. A two-byte operand can be specified by the "A" operator or the "B" operator. An operand preceded by an "A" and enclosed in parenthesis will be stored as a two-byte address. An operand preceded by a "B" and enclosed in parenthesis will also be stored as a two-byte address, but the low-order byte will be stored ahead of the high-order byte. Two-byte operands can be intermixed with single-byte operands in the operand list of the same DC statement.

Examples of DC instructions are as folows:

```
        DC      X'23'
        DC      ALPHA+1,21,'ABC'
LOC     DC      H(BETA),A(ALPHA+X'23')
        DC      'DON''T'
        DC      B(BETA)
        DC      A(X'3F21'),X'21'
```

DS

The DS pseudo-instruction is used to reserve memory space, but without initializing it to a specific value as with the DC pseudo-instruction.

The operand field of the DS statement contains the number of bytes of memory to reserve. This operand may be a constant symbol, or expression, but like the EQU and ORG statements, any symbols used in the operand must have been defined in a previous statement.

Example:

```
DATA        DS          25
```

PRNT ON — The PRNT ON and PRNT OFF pseudo instructions allow for selective listing of the assembler output. PRNT ON turns on the hard-copy listing device.

PRNT OFF — The PRNT OFF pseudo instruction turns off the hard copy listing device

3.5 Assembler Comments

If the first character in a statement is an "*" then that statement will be treated as a comment. Comments, of course, produce no effect on the assembly of the program. Examples of comment statements are as follows:

```
*
*THIS PROGRAM COMPUTES THE SQUARE
*ROOT OF A+B-2
*
```

The comment field can be extended beyond the end of the 40 character line by typing a "-" (hyphen) in position 40 of the line to be extended. The assembler will then concatenate that line with the line following. The maximum length allowed is the two lines.

3.6 Error Flags

The Assembler analyzes each input statement during pass 2 of the assembly process and is capable of detecting a large number of errors. The assembler uses a single character at the left of the listing line to flag a statement containing an error.

The error flags and their meanings are described as follows:

FLAG	ERROR DESCRIPTION
N	Error detected in label (name) field. The symbol in the label field contains more than eight characters or contains invalid characters.
O	Error detected in opcode field. The opcode was not recognized.

P Error detected in operand field. There are a number of syntax errors which cause this flag. They are:

1. Incorrect number of operands
2. Register operands not valid
3. Operand omitted when required
4. Unbalanced quotes on ASCII string
5. Improper syntax in expression
6. Missing or unbalanced parenthesis
7. Invalid operator
8. Relative branch beyond the allowed range (6800)

U Undefined symbol referenced in operand field.

R Invalid forward reference, or undefined symbol in ORG, EQU, or DS instructions. All valid references to a symbol which was incorrectly forward-referenced will also be flagged with an R, but may be ignored.

D The symbol in the location field has been defined more than once. Each occurrence of the duplicated symbol will be flagged.

I Invalid operand for Z-80 indexed instruction.

J Jump address is out of range for the Z-80.

A Attempt to load into protected memory in the QUICKRUN system. Use the "N" option to suppress automatic object loading.

4.0 Tape Monitor/Debugger and Microemulator Debugger

The Tape Operating System Debugger provides a wide range of debugging features. The Microemulator Debugger is made available when the optional MICROEMULATOR in-circuit emulator and EPROM programmer is ordered.

The Monitor/Debugger is bootstrap loaded from tape as previously described. The QUICKRUN Monitor/Debugger is invoked, from the other system functions, by typing ".M"

The FUTUREDATA Debugger program is composed of two principle elements: a program loader, and a program debugger.

The program loader reads programs into the microcomputer memory from a an input tape file. A dump routine is also available which performs the inverse functions; i.e., it writes data from memory out onto a tape output file and thereby creates a loadable object file.

The program debugger provides an interactive facility for testing programs. Once a program under test has been loaded into memory, the debugging commands can be used to display memory, store changes, and execute the program. Also, during execution, a program can be manually interrupted using the BREAK key. At this point, the Debugger receives control and saves the machine status. Debugging commands may then be entered to examine the state of the program. All internal registers and flag bits are displayed and may be changed. Execution can later be resumed from the point where the interrupt occurred.

4.1 Debugger Display Format

While the Debugger program is in control, the CRT always contains a formatted display of some portion of memory. The area of memory which is displayed is controlled by commands entered to the debugger.

The first two or three lines of the display are reserved for machine status. The machine status display includes the contents of the program counter, stack pointer, flags, and registers.

The following abbreviations are used in this display.

PC	program counter
SP	stack pointer
C	carry flag
Z	zero flag
S	sign flag
P	parity flag
D	decimal (half) carry
A	accumulator
B,C,D,E	scratch registers
H,L	H and L registers

8080A Symbols

PC	program counter
SP	stack pointer
C	carry flag
Z	zero flag
N	sign flag
V	overflow flag
I	interrupt flag
H	half-carry flag
ACCA	accumulator A
ACCB	accumulator B
X	index register

6800 Symbols

PC	program counter
S	stack pointer
C	carry flag
Z	zero flag
V	parity/overflow flag
S	sign flag
N	subtract flag
D	decimal (half) carry
A	accumulator
A'	accumulator (alt)
B,C,D,E	scratch registers
B',C',D',E'	scratch registers (alt)
H,L	H and L registers
H',L'	H and L registers (alt)
X,Y	X and Y index registers
V	Interrupt vector register
I	Machine instruction

Z-80 Symbols

This display shows the status of the machine at the point when

execution of a user program is interrupted by a manual or software interrupt. The machine status display is zeroed when execution of the Debugger begins. Breakpoint interrupts will be treated in greater detail in Section 4.4

In addition to the machine status display, a selected area of memory is displayed. Each line displays eight bytes which are each formatted into two hexadecimal digits. At the right of each line is an ASCII representation for those bytes which contain a displayable character code. Twenty lines of memory data are displayed which represent a 160-byte area of memory. At the left of each line is a 4-hexadecimal digit (16-bit) address. This is the address of the first data byte on the line. The actual address of any of the eight data bytes on a line can easily be obtained by counting from the left. To simplify this process, the display is always formatted so that the address of the leftmost byte is a multiple of 8 (i.e., ends in 0 or 8).

Following the address is a field which is used to flag write-protection. If the memory data displayed on any line is part of a write-protected page of memory, an "*" character is displayed after the memory address. This means that all bytes on the flagged line are protected.

When using the Microemulator Debugger and mapping memory into the user system, the symbol "+" next to the address indicates that this memory address is in the user system.

When the area of memory to be displayed is specified by giving the Debugger a particular memory address, this address is placed at the center line of the display and is marked by a ">" symbol to the left of the specified byte. This address at which the pointer is displayed is used as a default memory address in some of the Debugger commands. These commands are described in the following section. The bottom line is reserved for entering commands.

4.2 Debugger Commands

The Debugger prompts with a ">" symbol in the lower left corner of the display screen to signify that it awaits input. System commands are displayed following this symbol as they are typed. A "_" symbol serves as a cursor. The backspace and cancel (CTL X) keys are used to make corrections in the commands to the system.

The backspace deletes the last character typed and moves the cursor back. The cancel (CTL X) key deletes the whole command line so that a command may be completely retyped.

The end of the command parameters is signalled by a carriage-return key. The carriage return causes the Debugger to

execute the command.

If the Debugger detects an error in the format or content of a command, the message "SYNTAX" will be displayed at the bottom of the screen. The command can then be reentered.

In the command descriptions that follow, two types of numeric parameters will be encountered. Address parameters denoted by "n" and "m" are 16-bit values entered in hexadecimal. symbol to the left of the specified byte. This address at which the pointer is displayed is used as a default memory Data parameters, on the other hand, are denoted by "d" and are 8-bit values. Hexadecimal data values have a length of 1 or 2 digits. Again, if less than 2 digits are entered, a leading zero is assumed.

Lists of address or data parameters may be separated by a blank or a comma. Also, one or more blanks may separate a parameter field from the command.

The Debugger will evaluate address and data parameters that are expressions according to the following rules:

+	Indicates hexadecimal addition
-	Indicates hexadecimal subtraction
*	Indicates indirect addressing. This allows the user to point to a location in memory with the cursor and use the two subsequent bytes as an address. (note: for the 8080A and Z-80 the address is evaluated in low-high order; for the 6800 the address is evaluated in high-low order).
%	Indicates relative addressing. The display cursor will move to the address of the relative branch (for the Z-80 and 6800).
'D'	Indicates an ASCII parameter.

For example:

D1000+'A'*	Indicates "display memory" at the location whose address is stored at 1000+41 (41 is the hexadecimal value of A).
S 'D','A','T','A'	This will store the hexadecimal data 44 41 54 41 into the the four successive locations following the cursor.

When using the QUICKRUN software with a Microsystem/15 an additional operator is allowed.

#label	Indicates symbolic addressing using the assembler symbol table to look-up the address of "label".

The following table also includes the Microemulator commands. These, of course, will not function unless the Microemulator has been installed.

There are two sets of commands associated with the Microemulator. The first set of commands, the MODE SELECT, SINGLE STEP, USERMAP, NOINTERRUPT and TRACE commands, relate to the emulation and debugging functions. The second set of commands, VERIFY, INPUT, PROGRAM, and TEST CLEAR relate to EPROM programming.

The complete set of Debugger commands follows.

COMMAND	FUNCTION
D n	Display memory at location "n." Memory data is displayed on the CRT formatted in hexadecimal, eight bytes per line. The line containing the specified byte is positioned at the center of the screen and a ">" symbol marks the byte.
D+n	Move the display "n" bytes ahead.
D-n	Move the display "n" bytes back.
Dn+m	Display memory at the memory address which is the sum of the values "n" and "m."

Dn-m	Display memory at the memory address which is the difference of the values "n" and "m."
D$	Each time a new memory address is displayed, the previous address is saved. The "D$" command recalls this previous display address. This allows the operator to go back to the last display. When the store command is used, however, the end of the stored data replaces the saved display address. Following a store command, the "D$" command will move the display to the end of the stored data.
D*	The display indirect command moves the display to the address given by the two bytes in memory following the pointer. For the 8080A and Z-80 the data immediately following the pointer is treated as the low-order address byte. For the 6800 the data immediately following the pointer is treated as the high-order address byte. This command is useful for moving the display to the transfer address of a jump instruction.
D%	For the Z-80 and 6800, this form of the display command is used to display relative addresses. The byte following the pointer is treated as a two's-complement offset.
CTRL D	Moves the display pointer ahead one location. The RETURN key is not required after any of the CTRL commands.
CTRL S	Moves the display pointer back one location.
CTRL A	Moves the display pointer ahead eight locations (one line).
CTRL B	Moves the display pointer back eight locations.
CTRL D REPT	Scrolls ahead through memory one location at a time.

CTRL S REPT	Scrolls backwards through memory one location at a time.
CTRL A REPT	Scrolls forward through memory.
CTRL B REPT	Scrolls backwards through memory.
D#label	This indicates symbolic addressing using the assembler symbol table to look up the address. It is available only in the QUICKRUN tape system.
Sd	Store the data byte "d" at the location where the display pointer is positioned. The CRT display is updated to show the result of the store.
Sd, d,..., d	Store a series of data bytes in sequential memory locations beginning with the location where the display pointer is positioned. The address following the stored data is saved; the display can be moved to this location using the "D $" command. Since only the command line (40 characters) is allowed for the store command and data, it is occasionally the case that a long sequence of data to be stored cannot fit on the command line. The "D$" command is then useful to move the display ahead, after the first line of data is stored, so additional data may be stored.
En	Begin executing a program starting at location "n." The execute command causes the Debugger to load the registers and status flags with the values displayed as part of the machine status and then executes a jump to location "n" to transfer control to a user program. THE EXECUTE COMMAND WILL WORK PROPERLY ONLY IF THE DISPLAYED STACK POINTER HAS A VALID RAM MEMORY ADDRESS. If not, it can be reset by the "Z" command.
E	Begin execution at the location where the display pointer is positioned. As with the above form of the execute command, the registers and status flags are loaded

with the values displayed as part of the machine status before execution begins.

E$ — This form of the execute command is used after a program has been loaded from disk or after a breakpoint interrupt, and causes execution to begin at the location displayed for the program counter in the machine status display. The program counter display is set by the disk loader to the entry point of the loaded program and is set by the breakpoint interrupt handler to the address for resuming execution. Prior to beginning execution, the command restores the register and flag contents from the machine status save area as with the normal execute command.

E#label — Begin execution at the address of "label". (QUICKRUN only)

Z r=d — The zap commands provide a means for changing the values of the registers and flags displayed as part or the machine status. Altering these values has no immediate effect but changes the data loaded into the registers when execution of a program begins. For the 8080A the "r" may be any one of the valid register names A, B, C, D, E, H, L, and the data "d" is stored for later loading. For the 6800 the "r" may be any one of the valid register designations A, B, or X. For the Z-80 "r" may be any one of the valid register designations A, B, C, D, E, H, L, X, Y, A', B', C', D', E', H', or L'. That is, to set register B, the command.

```
Z B=d
```

is entered.

To reset the stack pointer, the following command is used:

```
Z S=n
```

In this case, "n" is a 1 to 4-digit hexadecimal memory address. Note that

since the machine status is stored in the stack, changing the stack pointer will also change all of the displayed machine status data.

For the 6800 and Z-80, the index registers (X and Y) are set with 16 bit values by the zap command.

Flags may be set by the command

Z F=d

All of the flags are set at once, and the bits of the data byte "d" have the following assignments:

	8080A	6800	Z-80
Bit 0 -	Carry	Carry	Carry
Bit 1 -	Unused	Overflow	Add/Subtract
Bit 2 -	Parity	Zero	Parity/Ovfl
Bit 3 -	Unused	Negative (sign)	Unused
Bit 4 -	Dec. Carry	Intr. Mask	Dec. Carry
Bit 5 -	Unused	Dec. Carry	Unused
Bit 6 -	Zero	Unused	Zero
Bit 7 -	Sign	Unused	Sign

Z r=d, r=d,...,r=d — Several registers may be set at once.

X — This command is used to change the write-protection status of the 1K memory block in which the display pointer is positioned. The "X" command inverts the protecton status from its current state. If the block is unprotected it becomes protected, or if it is protected it becomes unprotected. Protection may be set or reset only for whole 1K blocks on 1K boundaries. When memory is protected, an "*" appears to the right of the memory address in the memory display.

F d,d,d.... — The find command will search through memory (starting at the present cursor address) for the string of data "d,d,d....". If a match occurs the cursor will be left pointing at the starting address of the match string.

The data may be any combination of hexadecimal or ASCII constants.

F — This form of the find command will use the last specified argument (for the find command) as the test string.

BSn — The breakpoint set command allows up to 4 breakpoints to be set (n=1 to 4). This command will replace the byte at the cursor location with a software breakpoint (X'F7' for 8080A and Z-80; X'3F' for the 6800). When the user program encounters a breakpoint, control is returned to the debugger. All user registers are saved and automatically displayed. This aids in debugging and allows the programmer to resume execution with the processor status intact. When using the Microemulator the command BS0 is used to set the hardware breakpoint.

BDn — The breakpoint display command allows the user to locate the memory address for each breakpoint that has been set. Thus the breakpoints may be set, the program executed, memory examined and the cursor repositioned at the breakpoint. When using the Microemulator the command BD0 is used to display the address of the hardware breakpoint.

BR — When the cursor is positioned at at a location where a breakpoint has been set, then the breakpoint may be reset (the instruction restored) using the breakpoint reset command.

BC — The breakpoint clear command is used to inform the debugger that the breakpoint save area is to be restored to initial conditions. This is useful in the case when the object program is reloaded without resetting any or all of the existing breakpoints.

L — Loads object data from an object tape file into memory. Object tape must be in the input tape drive. When loading is complete, the "PC" display contains the

ADDENDUM TO

FUTUREDATA TAPE OPERATING SYSTEM USERS MANUAL

An,m

The allocate memory command allows the user of the Microsystem/15 to allocate the user portion of memory (18 kilobytes) between object code, source code, and assembler symbol table. Thus, the symbolic debugging facilities and instantaneous availability of monitor/debugger, editor and assembler may be extended to programs of much larger size.

The first parameter, "n", specifies the size of the object code work space (starting at X'ØØØ'). The object code workspace should be set to an integral one kilobyte boundary (X'Ø4ØØ', X'Ø8ØØ', X'ØCØØ' ...), since the memory protect feature operates on integral one kilobyte boundaries. If a non-integral region is specified, it will be expanded to fill up to the end of the next one kilobyte page. The Microemulator Trace area will occupy the top 286 locations in the object code area. The default user stack pointer will be set to the bottom of the trace area, unless respecified within the user program.

The second parameter "m", specifies the length of the assembler symbol table. Each symbol, within the user source program, requires 3 bytes plus the length of the symbol (in characters) itself. Thus, an average 5 character symbol will take 8 bytes of symbol table space, or put another way, each X'4ØØ' bytes will store about 128 symbols.

One note of caution is: Whenever the allocate command is used, it will reset all pointers into the Edit Buffer thus clearing the editor. The user should, therefore, be certain to dump the program in the Editor to tape prior to using the allocate command.

entry address.

L offset — Loads object data from the object tape file into memory, with the load address incremented by the hexadecimal value "offset". That is, as the data from the file is loaded, the value "offset" is added to the loading addresses specified in the file to compute the actual addresses into which the data is loaded. The "offset" parameter allows files to be loaded into arbitrary areas in memory. It is useful for loading data to be programmed into EPROM's when the EPROM address fall outside the system's RAM memory. Note that this command does not change any of the address references in the loaded data. A program origined at X'B000' and loaded into RAM at X'1000' cannot be executed because addresses within the program still reference addresses in the area around X'B000'. The command "L1000-B000" would cause a file origined at X'B000' to load starting at X'1000'.

W addr,addr,...addr — Writes data from memory into the object tape file. Data is written from memory starting with the location specified by the first address parameter and ending with second address parameter. Then data is written from the area between the next two address parameters, etc. If there is an even number of parameters, the first address specified is the entry point; otherwise the last parameter is used.

J function — Transfer control to another system function. (QUICKRUN only)

The following commands are supported by the Microemulator hardware and software.

COMMAND	FUNCTION
M parm(s)	The M command controls the mode of the Microemulator system. The M command allows a series of individual mode control functions. These are entered as series of parameters

following the M command. (i.e. M CD switches the clocks and DMA functions into the user system). These may be used individually or in combination, as required. There are various individual mode control parameters for the three processors (8080, Z-80 and 6800). These parameters are entered following the M command without blanks or commas to separate them. For example, the command to switch everything but DMA control on the 8080 is MCU. The parameters are as follows:

C - enable the user clock.

This causes the Microsystem to switch clocks from the internal host clock to the external user clock. (Note: if the user clock is not available, and the switch is attempted, it will cause the program to dump from the Microsystem.) The C parameter applies to 8080, Z-80 and 6800.

U - set user mode.

This enables the RESET, READY and INTERRUPT lines from the 8080 in the user system to activate the appropriate functions in the Microsystem. For the Z-80 this parameter is not used. For the 6800 the U parameter will switch the clocks and enable the memory map (see the U command).

D - enable user DMA

This causes the Microsystem to respond to external direct memory access requests and suspend processing as appropriate so that the external system may gain control of the system buss. The D parameter has the same function for the 8080, Z-80 and 6800. The HOLD and HLDA lines are switched for the 8080. The TSC, HALT and BA lines are switched for the 6800. The BUSRQ and BUSAK lines are switched for the Z-80.

I - enable user control lines

For the 6800 the I parameter allows the user system to control INTR-, NMI-, and RESET-. For the Z-80 the I parameter allows the user system to control INT-, NMI-, RESET-, WAIT-, and to respond to I/O reads and writes. (Note: in this mode the Z-80 will not generate any internal I/O requests).

M - enable user memory map

The M parameter enables the memory map function for the Z-80 (see the U command).

P - protect I/O requests

Disable the Microsystem I/O when running user programs in the 6800 Microsystem. This is especially useful in disabling the memory protect registers so that user programs cannot clear the protect registers and then clear memory. It is not required in the 8080 and Z-80 because of their separate I/O address space.

E - enable full emulation mode

Enable the clocks, RESET, READY and INTERRUPT lines and direct memory access for the 8080 emulator. Enable the clocks, NMI-, INTR-, RESET-, TSC, HALT-, DMA, and memory map for the 6800 emulator. Enable the clocks, NMI-, INT-, RESET-, WAIT-, DMA, I/O requests and memory map for the Z-80.

M

If no parameters are specified the emulator will return to their default modes as follows:

1) system is returned to the host clock;

2) the system will not respond to external reset, or control lines;

3) user direct memory access facilities are disabled.

The control mode will be altered after the M command is entered, upon pressing the RETURN key. The control mode will remain unchanged unless re-entered or if the "LOAD", "R" command sequence is used. This command sequence returns control to the Debugger through a vector in the bootstrap PROM. It will reset the control mode to the default values and disconnect the user system.

T

The T command causes a dump of the last 20 steps in the user program prior to an interrupt. The trace execution will begin where the display pointer is positioned. In the trace mode the processor is interrupted after every instruction.

The processor will execute approximately 1000 instructions per second in trace mode. The trace execution will terminate when: (1) a hardware or software breakpoint is encountered, or (2) the break key is struck. At this point a trace display will appear on the screen. The trace display provides the contents of the PC, the next instruction to be executed, all the registers, the flags, and the stack pointer for all 20 states. The trace display stores the 20 states immediatly preceeding the interrupt. This allows the designer to look backwards in time to determine what was happening prior to the interrupt. For the Z-80 only 9 states are displayed on the screen at any one time. However, the trace display can be rolled forwards with a CTRL A or backwards with a CTRL B. The standard dump display can be recalled by striking the return key.

The trace command causes the the debugger to load the registers and status flags with the values displayed as part of the machine status and then executes a jump to the requested location to transfer control to the user program.

T addr — Begin trace execution starting at location "addr". The termination of the trace execution is the same as above.

T# — Recall the trace display. This allows the programmer to use the standard debugger commands to display memory and then return to the trace display.

T$ — Begin trace execution at the present stored value of the program counter.

CTRL E — The control E command causes the single step execution of the user program. The designer may use the control E and the Repeat key on the keyboard to slow step through the program. By watching the cursor on the screen, the designer can trace the action of the program as it executes straight line code, jumps, loops, etc. The designer can further watch the register status display at the top of the screen and see the program counter, stack pointer, all of the registers and flags as they are updated during program operation. It should be apparent that the single step or repeat single step mode is useful

only in small segments of program.

U addr,map — For the 6800 and Z-80, the emulators allow for mapping of the processor address space between the Microsystem (ROM-PROM Simulator) and the user system. The USERMAP command treats the address space as 256 segments of 256 bytes. These can arbitrarily be allocated to the host or user system.

The command transfers the memory space specified by "mask" starting at "addr" into the user system.

Each user map begins on a 2K boundary. Each bit in the mask will switch 256 bytes into the user system. The bits are individually "or'ed" to form the mask. Thus U 1800,0F would switch the memory space from X'1800' to X'1BFF' into the user system. If "addr" is not on a 2K boundary then the message "NOT ON 2K BOUNDARY" will be displayed.

U,mask — Transfer the memory space specified by "mask" starting at the present cursor location into the user system. If the cursor is not positioned on a 2K boundary the message "NOT ON 2K BOUNDARY" will be displayed.

N addr — Begin user program execution at location "addr" with interrupts disabled. This command is useful in debugging user systems, since it allows all the control lines except interrupt to function. (Note: for the 6800 and Z-80 the NMI interrupt function is not disabled).

N — Begin user program execution at the cursor location with interrupts disabled.

N$ — Begin program execution at the location displayed for the program counter in the machine status display with interrupts disabled.

I — The I command (input) will input the contents of a 2704 or 2708 EPROM from the EPROM programmer. "N" bytes will be loaded starting at the cursor location. "N" is determined by the last length parameter (note the length parameter may be set any of the EPROM commands).

I addr,length — Causes the contents of the EPROM to be loaded into

memory starting at "addr". The number of bytes loaded is equal to "length".

I addr — If length is unspecified then the last value entered will be used. The default length is 1024 bytes.

I ,length — Causes the contents of the EPROM to be loaded into memory starting at the cursor address. The number of bytes loaded is equal to "length".

C — The C command (clear) will test the contents of a 2704 or 2708 EPROM to determine if it is empty (all locations X'FF'). The number of locations checked is determined by the last length parameter entered. If the EPROM is clear the system will respond with the message "EPROM IS EMPTY". If the EPROM is not clear the system will respond with the message "NOT EMPTY AT addr"; where "addr" is first non-clear location.

C length — This will test the first "length" bytes for the empty condition.

P — The P command (program) will program the EPROM in the EPROM programmer socket from Microsystem memory. "N" bytes will be programmed starting at the cursor location. "N" is determined by the last length parameter entered.

The EPROM programmer will apply 25% of the manufacturers recommended number of pulses and then test for correct data. If the data is correct the remaining 75% of the programming pulses will be applied. Otherwise the message "MISMATCH AT addr" will appear; where "addr" is the first address in the EPROM with erroneous data.

P addr,length — This will program the EPROM with the contents of memory starting at "addr". The number of bytes programmed will be equal to "length".

P addr — If length is unspecified then the number of bytes programmed will correspond to the last length specification.

P ,length — This will program the EPROM with the contents of memory starting at the cursor address. The number of bytes programmed will be equal to "length".

V — The V command (verify) will compare the EPROM in the EPROM programmer socket against the contents of the Microsystem memory. "N" bytes will compared starting at the cursor location. "N" is determined by the last length parameter entered.

If there is a mismatch the message "MISMATCH AT addr" will appear; where "addr" is the first address in the EPROM that has erroneous data.

V addr,length — This will compare the contents of the EPROM with the contents of the Microsystem memory starting at "addr". The number of bytes compared will be equal to "length".

V ,length — This will compare the EPROM with the contents of the Microsystem memory starting at the cursor location. The number of bytes compared will be equal to "length".

4.3 DEBUGGING FACILITIES

In Section 4.2, the Debugger commands for displaying and storing into memory were described in some detail. These provide the most important and useful debugging tools. Once a program under test has been loaded into memory from tape, execution of the program can be attempted. If the program does not perform properly, control can be returned to the Debugger and memory can be examined for a clue to the problem. If an error is discovered, a patch can be inserted and execution can be restarted. The interactive nature of this process greatly reduces debugging time.

When a program is malfunctioning, control can be returned to the Debugger in two standard ways. First, if interrupts are enabled, as indicated by the ENABLED status indicator on the keyboard, (for the 6800 and Z-80 the BREAK Key Interrupt is always enabled) then the BREAK key may be used to return to the Debugger. If this means of aborting execution is used, the Debugger will save the machine status at the moment of interrupt. The program counter, stack pointer, registers and flags are all saved and displayed. The displayed values may be analyzed and if necessary changed using the "Z" command. When execution is started again the displayed values are reloaded prior to transfer of control back to the user program. The "E$" command resumes execution at the location displayed for the program counter. This is the location of the next instruction to be executed when the interupt occurred.

The second standard means of returning control to the Debugger from a malfunctioning program is by using the RESET key. This is

not as clean as using the BREAK key and should be used only when the interrupts are disabled and cannot be reenabled. When system reset is used, the machine status cannot be reliably saved, and consequently the machine status display will not have accurate information. However, control is returned to the Debugger and all memory data is secure.

A software breakpoint instruction can be patched into a program, (using the Breakpoint Set command) and will have the same effect as if the interupt key were pressed. For the 8080A and Z-80 the instruction which causes a breakpoint and transfers control to the Debugger is an RST 6. RST 7 which has an easy-to-remember hexadecimal code of FF, will also transfer control to the Debugger for the 8080A and Z-80. For the 6800 the software interrupt is X'3F'. When a software interrupt instruction is executed, control is passed to the Debugger which saves the complete machine status as for a manual interrupt.

For the Z-80 and 8080A an additional benefit is available, since all areas of nonexistent memory are read as X'FF', such memory, in effect, is filled with RST 7 instructions. An erroneous jump into nonexistent memory will , therefore, cause a breakpoint back to the Debugger.

One further method exists for returning control to the debugger. This is via a vector in the bootstrap PROM. A user error can clobber the reset and/or breakpoint vectors. (This is especially true with the 6800 where the instruct X'FF, X'FF', X'FF will store the contents of the index register at location X'FFFF.) In this case the BRK and/or RESET keys may not return control to the debugger. If this happens the user should press the "LOAD" key and type "R" ("LOAD" key, "BRK" key and "6" for the 8080). This will return control to the debugger via the bootstrap PROM.

In a computer system with an elaborate front panel, it is often convenient to single-step a program to watch how it executes. However, this is a time-consuming and tedious way to find a problem, especially in a long loop.

A bug can generally be isolated much more quickly using breakpoints. The breakpoint instruction is set at a suspected point in the program and execution is begun. When the breakpoint is reached, control goes to the Debugger where machine status and memory can be examined. If that point is not reached (i.e., the breakpoint never occurs), this indicates that the program is failing earlier, and another breakpoint can be set. It is very rapid process to zero-in on a bug using this technique.

Breakpoint instructions are most easily inserted during debugging by using the breakpoint set command. When it is necessary to

remove the breakpoint, the original instruction must be restored in place of the breakpoint instruction using the breakpoint reset command. The breakpoint display command provides a convienent method for locating any breakpoints that have been set. Also, when the breakpoint is executed, the program counter will be left pointing at the following instructions.

If the breakpoint is removed, execution should be restarted using the "E" command. The "E" command will restore the status and begin execution with the instruction just restored.

5.0 Operating the Microsystem

The Microsystem is powered-on by the switch on the front panel. When the CRT display and the tape units are plugged into the convenience outlets at the back for the system, these devices are also turned on by the main power switch.

The power-on logic causes a master system reset, and then automatically invokes the bootstrap loader routine. The procedures in the following section can then be used to load any of the Microsystem tapes.

5.1 The Tape Bootstrap Loader

The tape bootstrap loader routine is permanently resident in read-only memory and is invoked when the LOAD button is pressed or following power-on. The tape bootstrap loader reads programs from cassette tape into the microcomputer memory, and when loading is complete begins execution. The three standard software systems (monitor, editor, and assembler) can all be loaded by the bootstrap routine.

The procedure for loading Microsystem programs using the bootstrap is as follows. The program tape is placed in cassette unit 1, and the PLAY key is depressed to ready the unit for reading. When the LOAD button is pressed, (in the case of the 8080A) the tape will begin moving. For the 6800 and Z-80 systems the "T" key must be pressed following the LOAD button or initial power-on. In the case of an initial load following power-on, the bootstrap is already started and the LOAD button need not be pressed. Consequently, as soon as the cassette is inserted and the READ key is pressed, reading of the cassette begins.

After the cassette tape leader has been passed (approximately ten seconds), the ENABLED status indicator will periodically blink to indicate that data is being read into memory. When loading is completed, the program that has been loaded will begin executing and a REWIND message will be displayed on the CRT. The program tape should be rewound by pressing the REWIND key, waiting for the tape to rewind, and then pushing the STOP key. Finally, the user must signal the Microsystem that the rewind is complete by pressing carriage-return. At this point, the program that has been loaded is ready for use.

If a checksum error is detected during the bootstrap operation, this condition is indicated by halting execution of the bootstrap program. The halt condition is indicated when the RUN indicator goes dark. Rewind the tape and initiate loading again by pressing the LOAD button.

5.2 Restarting the Monitor/Debugger

The Microsystem monitor/debugger is loaded by the bootstrap routine using the above procedure. When the monitor/debugger begins execution, it write-protects the memory area in which it resides. This prevents the interactive debugger from being clobbered by a test program that malfunctions and steps through memory.

Normally, for the 8080A and Z-80 a jump instruction is also loaded at location zero, which causes the monitor/debugger to receive control when the RESET button restarts the execution from location zero. For the 6800 the address of reset entry for the monitor/debugger loaded in locations X'FFFE-F'. The 6800 then restarts execution at this address. However, since these areas of memory are not reserved exclusively for monitor/debugger use, they are not write-protected and can occasionally be clobbered by a test program. Thus it sometimes occurs that it is impossible to restart the monitor/debugger with the RESET button even though the monitor/debugger is intact in memory.

An escape function has been built into the bootstrap so that control can be returned to the monitor/debugger in the situation just described. The sequence for restarting the monitor/debugger is as follows. Press the LOAD button to initiate the bootstrap. Then the key sequence "BREAK" and "6", (or LOAD and "R" in the case of the 6800 and Z-80), will transfer control to the monitor. After the monitor/debugger is executing, it will be necessary to restore the jump instruction at zero (or the reset vector at X'FFFE-F') if the RESET button is to have an effect.

5.3 The "RESET" Button

The RESET button causes a master system reset which has the following effects. All memory protect bits are reset so that memory is unprotected, and all I/O device mask bits are reset so all interrupts are masked. Then the master reset of the CPU interrupt enabled flip-flop and restarts execution at location zero.

The system reset is used to restore the microcomputer to an "initial" state. It does not save any machine status, and consequently the values of the CPU registers are unpredictable when execution resumes.

A master system reset as described above is also generated when the LOAD button is used.

5.4 Cassette Tapes

The cassette tape used on the Microsystem tape units may be ordinary audio tapes. While any brand of tape should work satisfactorily, the low-cost 3M Highlander cassettes have proven to be especially reliable and durable.

Cassette tapes may be write-protected (i.e., writing is prohibited) by breaking out the plastic tab at the back of the cassette. Data can be recorded on either "side" of the tape. Consequently two write-protect tabs are provided so that each side can be separately protected. When looking at the back of the cassette, the tab at the right is associated with the top "side" of the cassette.

5.5 Cassette Tape Units

The cassette tape units are standard audio cassette recorder/players. The units are unmodified and data encoded at audio frequencies is fed into the microphone input and recovered from the monitor output. Tape motion control is provided using the remote switch input. The microcomputer is therefore able to turn the tape on and off as required.

The cassette unit is prepared for reading a tape by placing the tape into the unit and pressing the key labeled READ. The cassette may not begin moving until the program actually begins reading.

Once the tape is read, it should always be rewound. Rewinding is accomplished by first pressing the STOP key which causes the READ key to pop up, and then pushing the REWIND key. Because the cassette motor is off when not reading, the REWIND key is only effective when the program has left the unit turned on for rewind. When the tape has completed rewinding, press the STOP key and the EJECT key to remove the cassette. Finally, the software requires that the RETURN key be pressed to indicate that the rewind sequence is complete.

The cassette unit is prepared for writing a tape by placing the tape into the unit and pressing the WRITE and READ keys simultaneously. If the cassette has been write-protected, the WRITE key is mechanically interlocked and cannot be depressed. Rewinding after writing follows the same sequence as described above.

5.6 Error Recovery During Loading

When loading a tape using the monitor/debugger "L" command, a checksum error is indicated by the "ERROR..RETRY" message. If

this message appears, rewind the tape, and then restore the unit to PLAY mode. Indicate to the monitor/debugger that the backspacing has been performed by hitting the RETURN key and loading will be attempted again. If the message appears more than once, it is very likely that the error is permanent, and was generated by a transient during writing, a kink in the tape, or the like. To exit from the loader if loading is aborted, use the RESET button.

Because such errors occur infrequently, and to minimize the loading time, load tapes are written without redundancy. It is recommended, therefore, that backup copies be made for important tapes, especially the system tapes. Procedures for copying these tapes are given in the section 1.2.5. Such tapes which are used frequently run a higher risk of being damaged or broken.

5.7 Crt Adjustments

Brightness of the display can be adjusted using the CONTRAST control only, on the rear of the CRT.

5.8 Machine Status Indicators

Two light-emitting-diode (LED) displays are provided on the keyboard to show hardware status. The first LED is labeled RUN and is lit when the CPU is executing instructions. This display will go dark when the CPU executes a halt instruction. When the CPU halts, execution can be restored by the BREAK or RESET keys only.

The second LED is labeled ENABLED and shows the status of the CPU's interrupt enable/disable status flip-flop for the 8080A. When illuminated, interrupts are enabled, and when dark interrupts are disabled. For the 6800 and Z-80 systems this LED is used only as visual feedback during tape operations.

6.0 Debugging Facilities

In Section 2, the monitor/debugger commands for displaying and storing into memory were described in some detail. These provide the most important and useful debugging tools. Once a program under test has been loaded into memory from tape, execution of the program can be attempted. If the program does not perform properly, control can be returned to the monitor/debugger and memory can be examined for a clue to the problem. If an error is discovered, a patch can be inserted and execution can be restarted. The interactive nature of this process greatly reduces debugging time.

6.1 Manual Interrupts

When a program is malfunctioning, control can be returned to the monitor/debugger in two standard ways. First, if interrupts are enabled, as indicated by the ENABLED status indicator on the keyboard, (for the 6800 and Z-80 the BREAK Key Interrupt is always enabled) then the BREAK key may be used to return to the monitor/debugger. If this means of aborting execution is used, the monitor/debugger will save the machine status at the moment of interrupt. The program counter, stack pointer, registers and flags are all saved and displayed. The displayed values may be analyzed and if necessary changed using the "Z" command. When execution is started again the displayed values are reloaded prior to transfer of control back to the user program. The "E$" command resumes execution at the location displayed for the program counter. This is the location of the next instruction to be executed when the interupt occurred.

The second standard means of returning control to the monitor/debugger from a malfunctioning program is by using the RESET key. This is not as clean as using the BREAK key and should be used only when the interrupts are disabled and cannot be reenabled. When system reset is used, the machine status cannot be reliably saved, and consequently the machine status display will not have accurate information. However, control is returned to the monitor/debugger and all memory data is secure.

6.2 Software Interrupts

A software breakpoint instruction can be patched into a program, (using the Breakpoint Set command) and will have the same effect as if the interupt key were pressed. For the 8080A and Z-80 the instruction which causes a breakpoint and transfers control to the monitor/debugger is an RST 6. RST 7 which has an easy-to-remember hexadecimal code of FF, will also transfer control to the monitor/debugger for the 8080A and Z-80. For the 6800 the software interrupt is X'3F'. When a software interrupt

instruction is executed, control is passed to the monitor/debugger which saves the complete machine status as for a manual interrupt.

For the Z-80 and 8080A an additional benefit is available, since all areas of nonexistent memory are read as X'FF', such memory, in effect, is filled with RST 7 instructions. An erroneous jump into nonexistent memory will , therefore, cause a breakpoint back to the monitor.

6.3 Finding a Bug Using Software Breakpoints

In a computer system with an elaborate front panel, it is often convenient to single-step a program to watch how it executes. However, this is a time-consuming and tedious way to find a problem, especially in a long loop.

A bug can generally be isolated much more quickly using breakpoints. The breakpoint instruction is set at a suspected point in the program and execution is begun. When the breakpoint is reached, control goes to the monitor/debugger where machine status and memory can be examined. If that point is not reached (i.e., the breakpoint never occurs), this indicates that the program is failing earlier, and another breakpoint can be set. It is very rapid process to zero-in on a bug using this technique.

Breakpoint instructions are most easily inserted during debugging by using the breakpoint set command. When it is necessary to remove the breakpoint, the original instruction must be restored in place of the breakpoint instruction using the breakpoint reset command. The breakpoint display command provides a convienent method for locating any breakpoints that have been set. Also, when the breakpoint is executed, the program counter will be left pointing at the following instructions.

If the breakpoint is removed, execution should be restarted using the "E" command. The "E" command will restore the status and begin execution with the instruction just restored.

7.0 Programming the Microsystem

7.1 Memory Configuration

The Microsystem/10 comes with an 8K byte read/write memory module. This memory normally occupies the low-order memory addresses X'0000' to X'1FFF'. The bootstrap loader makes use of the high-order 8K module area, locations X'E000' to X'FFFF' for the Z-80 and 8080A. For the 6800 the bootstrap loader makes use of the address area from X'C000' to X'CFFF'. While the bootstrap only occupies a 256-byte PROM, this PROM responds at all addresses in the high-order 8K area. There are no restrictions on referencing memory locations which are not implemented. When such a location is read, the value X'FF' will be obtained.

The monitor/debugger occupies 5K bytes of memory including I/O CRT display area (X'1C00' to X'1FFF'). For the 8080A and Z-80 this area is mapped into the corresponding area (X'CC00' to X'DFFF'). For the 6800 this area is mapped into the corresponding area (X'EC00' to X'FFFF'). The mapping for the 6800 allows the 6800 vectors (at X'FFF8' to X'FFFF') to be in RAM.

For the 8080A and Z-80 the I/O instructions refer to a separate address space. For the 6800 the I/O instructions refer to part of the main memory address space. The Microsystem I/O ports in the Mod 6800 are located in the address space X'D000' to X'DFFF'. For the Mod 6800 the I/O addresses are not completely decoded. Thus the Microsystem bus allows for 256 I/O locations even though the Mod 6800 uses up 4096 address locations.

7.2 Memory Address Mapping

Additional 8K memory modules can be installed in the machine to provide more accessible memory. It is desirable, however, to always run the monitor/debugger in the highest-order 5K bytes of available memory. In order to make this easy, an address mapping feature has been implemented. The monitor/debugger is actually assembled to run in the highest-order 5K area available ahead of the bootstrap. For the 8080A and Z-80 This area extends from X'CC00' to X'DFFF'. These addresses are mapped into the highest-order memory actually implemented in the machine using the memory address selection strapping option. In the Microsystem/10 the addresses X'CC00' to X'DFFF' are mapped into X'C00' to X'1FFF'. If, however, a second 8K module is installed, the memory is then strapped so that the addresses X'CC00' to X'DFFF' are mapped into X'2C00 to X'3FFF'. The monitor/debugger then, in effect, runs in the high-order 5K bytes of the second memory module.

The same strapping conventions hold for the Mod 6800. However, in

this case the mapping is from the high-order 5K bytes of the highest memory module into X'EC00' to X'FFFF'.

While the monitor/debugger write-protects the first 4K bytes of its area, which contains code, the last 1K is used for the CRT display and consequently is not write-protected. This 1K page is in the area X'DC00' to X'DFFF' and is mapped into X'1C00' to X'1FFF' in the Microsystem/10. For the Mod 6800 the area from X'FC00' to X'FFFF' is used for the screen display and is mapped into X'1C00' to X'1FFF' in the Microsystem/10. Though the monitor/debugger cannot be clobbered by storing data in this area, the results of doing so are unpredictable, especially when using the store (S) command. It is, however, acceptable for a program using the CRT display directly to store into the first 960 bytes of this area (see Section 9.4).

7.3 Monitor/Debugger Interrupt Vector Usage

The 8080A and Z-80 have a common 8-level interrupt structure. The 6800 has a 4-level interrupt structure including Restart and Software interrupt.

7.3.1 8080A and Z-80 Interrupt Vector Usage

The low-order area of memory from X'0000' to X'003F' is reserved for the 8080A and Z-80 interrupt vectors. While the assignment of vector addresses to I/O device interrupts is detailed in Section 7, there are three vector addresses which are used for monitor/debugger operations. These addresses, and the jump instructions required for monitor/debugger operations, are as follows:

Vector Address	Data	Instruction
X'0000'	C3 FD DB	JMP X'DBFD'
X'0020'	C3 FA DB	JMP X'DBFA'
X'0030'	C3 FA DB	JMP X'DBFA'
X'0038'	C3 FA DB	JMP X'DBFA'
X'0066'	C3 FA DB	JMP X'DBFA' (Z-80 only)

The jump instruction at location X'0000' allows the monitor/debugger to be restarted using the RESET button. The jumps at locations X'0020' and X'0030' are used for manual BREAK key interrupts and breakpoint instruction handling respectively.

While these locations may be changed in conjunction with running a program that uses these vector locations, the monitor/debugger features associated with the location changed will be disabled. Restoring the data as shown above will reinstate the

monitor/debugger functions.

7.3.2 6800 Interrupt Vector Usage

The 6800 version of the monitor/debugger sets up the 6800 vectors at locations X'FFF8' to X'FFFF'. These addresses represent the entry locations in the monitor/debugger for the RESET key, the BREAK key and software interrupt. The addresses and vectors are as follows:

Address	Vector	Function
X'FFF8'	X'FBFA'	Unused
X'FFFA'	X'FBFA'	Software Interrupt
X'FFFC'	X'FBFA'	Break Key
X'FFFE'	X'FBFD'	Reset Key

7.4 CPU Stack

When execution of a program is interrupted by the BREAK key or by executing a breakpoint instruction, control is transferred to the monitor/debugger which stores the machine status and registers on the stack, and then saves the stack pointer address. Unless a valid stack pointer was set up during the execution of the user program, the status saved using the stack may not be valid. In any case, since the monitor/debugger uses a stack which it sets up in its own temporary storage area, it will execute properly. However, the machine status display will not show valid data unless the stack pointer was properly set up at the time of the interrupt.

The execute command (E) will also not work properly unless a valid user stack is available. The execute command stores the starting address in the stack and then reloads the registers and executes a return instruction. The return transfers control to the execute address stored in the stack. If the stack pointer is not valid, especially if it points at nonimplemented memory or PROM, then the execute command will fail and generate the message "SP NOT IN RAM".

The stack pointer can always be reset by the "Z S=n" command. When this is done, the machine status display will change since the monitor/debugger generates the status display from the data in the stack.

When the monitor/debugger is first executed or when it is restarted using the RESET button, the user stack is set up just ahead of the monitor/debugger at X'CC00' (or X'EC00' for the 6800). This address is, of course, mapped to a real memory address; in the Microsystem/10 it is X'0C00'. When a program is

executed using the "E" command, this value is loaded into the stack pointer register. Either this value may be used without modification or the user program can reload the stack pointer to a required value.

7.5 Generating Bootstrap Loadable Tapes

Load tapes written by the monitor/debugger using the write (W) command or by the assembler can be loaded by the bootstrap loader This provides the capability for creating stand-alone software which runs directly on the Microsystem without the monitor/debugger. When generating a tape to be loaded by the bootstrap, an entry point MUST be specified, since when the bootstrap finishes loading, it jumps to the entry point to execute the program.

In addition, when the program is loaded, the load tape unit is left on for rewind. The program receiving control should first display a rewind message and then wait for the rewind before continuing execution.

The user should note that it is the responsibility of the user program to:

Initialize the Stack Pointer.

Set up the memory protect registers.

Set up the interrupt vector addresses or instructions.

Generate a rewind message and call the rewind routine.

Select the page for the CRT display.

These functions are normally performed by the monitor/debugger during initialization. However, the monitor/debugger initialization routines are not generally included in a user generated bootstrap loadable tape.

8.0 Monitor I/O Service Routines

In addition to the interactive facilities, the FUTUREDATA monitor also includes a number of service subroutines which greatly simplify input/output operations to the standard peripheral devices.

I/O service subroutines may be invoked by executing a CALL instruction to the appropriate memory location. Table 8.0 shows the entry point addresses of the monitor I/O services.

I/O subroutines are provided for keyboard input, CRT output, cassette read/write, and EIA send/receive (the EIA and CRT routines are supplied separately). These routines have convenient calling conventions for initiating I/O to the devices. The routines interpret the requests and perform the detailed, low-level I/O instructions to accomplish the requested operations.

8080A and Z-80 Address	6800 Address	Service Function
DBFA	FBFA	Breakpoint entry
DBFD	FBFD	Reset entry
DBE2	FBE2	Keyboard input
DBE5	FBE5	CRT output
DBE8	FBE8	Cassette read
DBEB	FBEB	Cassette write
DBEE	FBEE	Cassette read retry
DBF1	FBF1	Cassette rewind
DBF4	FBF4	EIA receive
DBF7	FBF7	EIA send

Table 8.0 SERVICE ROUTINE VECTOR ADDRESSES

Note: Most versions of the Monitor/Debugger, Emulator Debugger and QUICKRUN Debugger do not have the CRT and EIA routines included. This has been done to reduce the system program size by deleting functions only rarely used. The user who needs these routines may obtain them in source form by ordering the "IO-SUBR" tape.

8.1 Keyboard Input

The keyboard input requires no parameters. When called, it waits for a key to be pressed on the keyboard. When a key is received, it reads the ASCII code into the accumulator (A) register prior to returning to the calling program. Only the accummulator and flags are modified by this routine; all other registers are unmodified.

8.2 CRT Output

The CRT display may be addressed directly as will be described in

Section 9.4. However, a much easier way of outputting to the display is by using the CRT output service routine.

The CRT output simulates the functions of a CRT terminal. That is, a cursor " " is displayed which denotes the location where the next character will be stored. When the character is stored, the cursor is advanced. The cursor moves from the last character of the line to the first character of the following line. From the last character of the last line, it moves to the first character of the first line.

Several control commands are interpreted as follows:

Character	Function
Carriage-return	Moves the cursor to the left of the current line.
Line-feed	Moves the cursor down one line.
Backspace	Moves the cursor back one space.
Control P (DLE)	Clears the screen and leaves the cursor at the top left.

The line below the cursor is displayed as a horizontal dividing line of "-" characters. This dividing line separates the data most recently displayed just above it from the oldest data just below it. As new data is received and entered into the display, the dividing line is moved down. At the bottom of the screen it wraps around to the top. Data will remain on the screen, therefore, wntil it is eventually written over on the next cycle.

The ASCII representation of the character to be displayed is loaded into the accumulator prior to calling the CRT display routine. Before beginning a display, the cursor location should be initialized to the top left by outputting a clear (DLE=X'10') character.

All registers and flags saved and restored by the CRT output routine. For the 6800 the CRT routine does not save the flags.

8.3 Cassette Input/Output

The cassette I/O service routines provide for reading, writing,

and rewinding cassette tapes. Data on cassette tape is formatted as fixed-length records separated by inter-record gaps. The gaps are long enough so that the tape can be stopped at the end of a record and then restarted for reading the next record. The tape I/O routines provide for reading or writing a record at a time and automatically turn the tape units on and off.

The cassette tapes themselves have a short length of a plastic leader at the beginning which must be bypassed before reading or writing begins. A flag bit indicates the first read or write, and the I/O routines then automatically space past the leader.

For increased reliability, the I/O routines provide for redundant recording of data records. That is, any number of copies of each record can be specified and will automatically be written by the cassette write routine. During reading, if an error is detected, the read routine will try to read the record again. If another copy is available, it will be possible to recover from the error. If the read routine cannot find another copy, it will return with an error code.

The error recovery scheme makes use of sequence numbers which are recorded with the data of each record. Thus when an error occurs, subsequent data can be automatically identified as another copy of the desired record. Furthermore, if the tape is manually backed up for retry, the sequence numbers are again used to identify the desired record.

Parameters are passed to the I/O routines in the form of a 7-byte write-control block (WCB) and a 6-byte read-control block (RCB). The address of the control block must be loaded into the H and L registers, or the index register for the 6800 CPU, prior to calling the read or write subroutines.

8.3.1 Cassette Write Routine

The write-control block contains the fields shown in Table 8.1. These fields represent seven sequential bytes in memory and provide the cassette write routine with all the information necessary to process the request. For the first write, bit 0 (low-order) of WCBFLAG should be set to 1 so that a leader is written. This flag bit is automatically reset to 0 after the leader is written. Bits 1 and 2 of WCBFLAG should be used in cases where sequences of records are to be written without inter-record gaps. When bit 1 is set to 1, the write routine leaves the cassette unit on. Normally the unit is turned on before writing a record and off afterwards. In conjunction with bit-2, which suppresses the writing of an inter-record gap, multiple records can be formatted without gaps. The gap is normally necessary ahead of the data to give the tape unit enough

time to come up to speed. Thus if the gap is suppressed, the tape must have been left on after the previous write.

Field	Bytes	Function
WCBFLAG	1	Bits in this field request special functions as follows.
		Bit 0 --- Flags write of first record.
		Bit 1 --- Leaves tape unit on after write.
		Bit 2 --- Suppresses writing an inter-record gap ahead of the data.
		Bits 3-7 --- Unused-should be zero.
WCBADR	2	Address of data block to be written. Address is stored low-order byte, followed by high-order byte.
WCBLEN	2	Length of data block to be written. Length is stored as low-order byte, followed by high-order byte.
WCBUNIT	1	Write unit (1 or 2).
WCBCOPY	1	Number of copies of data to be written.

Table 8.1 Write Control Block

The WCBADR field of the WCB specifies the memory address of the beginning of the data block to be written to tape. The length of this block is specified by WCBLEN. Both WCBADR and WCBLEN are 2-byte (16-bit) fields with the low-order byte stored before the high-order byte. The WCBUNIT field specifies the tape unit to be addressed, either 1 or 2. Finally, the WCBCOPY field specifies the number of copies of the record to be written. For maximum data density without automatic error recovery, one copy can be specified. For practically all other applications, two copies is adequate. Two copies provides a hard error rate of about one in 1xE12 bits.

All registers and flags are saved and restored by the cassette write routine. The 6800 version does not save the flags.

8.3.2 Cassette Read Routine

The read-control block contains the fields shown in Table 8.2. These fields represent six sequential bytes in memory and provide

the read routine with all the information necessary to process the request. As with the write routine, the control block address must be loaded into H and L, or the index register for the 6800, prior to calling. For the first read of a tape, bit 0 of RCBFLAG should be set to 1. This causes the read routine to reset the record sequence number counter to zero in order begin reading a new tape. The bit is automatically reset to 0 after the first read. Bit 1 of RCBFLAG specifies that the cassette unit is to remain on after the read. This allows for reading of tapes where no inter-record gaps were written.

Normally, the read routine expects the records to have sequential record numbers, beginning with zero. However, under some conditions it is useful to be able to suppress the checking of sequential numbers and to reset the sequence number counter to the number of the next record read. Bit 2 of RCBFLAG performs this function. Once a record is read and the sequence counter is reset, zeroing bit 2 will allow for normal reading of sequential records.

Field	Bytes	Function
RCBFLAG	1	Bits of this field request special functions as follows:
		Bit 0 --- Flags read of first record.
		Bit 1 --- Leaves tape unit on after read.
		Bit 2 --- Accepts record with any sequence number.
		Bits 3-7 --- Unused-should be zero.
RCBADR	2	Address of data block into which data from tape is to be placed. Address is stored as low-order byte, followed by high-order byte.
RCBLEN	2	Length of data block. Length is stored as low-order byte followed by high-order byte.
RCBUNIT	1	Read unit (1 or 2).

Table 8.2 Read Control Block

Similiar to write, RCBADR and RCBLEN fields specify the address and length of the memory area into which the record is to be read. Both fields are two bytes (16 bits) and the values are stored with the low-order byte first. RCBUNIT specifies the read unit, either 1 or 2.

Upon return from the read routine, the accumulator (A) register contains a completion code: zero for success, nonzero for error. Several types of errors are classified by the value of the return code as follows:

Code	Meaning
1	Checksum error, normal retry should be attempted.
2	Data on tape is not recognizable, and retry is not likely to be successful. This code results when the data on the tape has a different record size than specified by the RCB.
3	No data has been read from the tape for an unusual length of time. Possibly a blank tape is being read.

All registers except for the accumulator and flag bits are saved and restored by the cassette read routine. For the 6800 the index register is saved.

8.3.3 Cassette Read Retry Routine

If a type 1 error is detected, recovery should be attempted. After the operator manually rewinds the tape for several seconds, the cassette read retry routine can be called. This call requires no parameters and causes the read routine to search for the record by sequence number, and then attempt to reread the data. The retry routine will also return a completion code. An error detected after retry probably means an unrecoverable error. The bad record can be skipped by simply calling the regular read routine again.

The retry routine saves and restores all registers except for the accumulator and the flag bits. The 6800 saves only the index register.

8.3.4 Cassette Rewind Routine

The rewind routine is provided for turning the cassette units on and off for rewinding. It does not perform message display on the CRT, this is left to the calling program. When called, the rewind routine turns on the selected tape for rewinding and then waits for operaor acknowledgment of completion. This is done by pressing the carriage-return key on the keyboard. When the signal is received that the rewind has finished, the unit is turned off.

The rewind requires the address of a RCB or WCB in the H and L registers, or the index register for the 6800. The control block specifies the tape unit to be rewound. Also, as a convenience, the routine sets bit 0 of the control block to prepare for reading or writing a new tape. The assumption is made that since the current tape has been rewound, a new tape will follow.

All registers and flags are saved and restored by the rewind routine. For the 6800 the flags are not saved.

8.4 EIA Input/Output

The EIA send/receive routines provide for sending and receiving serial data via the two EIA RS-232 ports.

Data is passed to and received for the EIA routines one character (byte) at a time. Parameters passed to the routines specify word length, transmission rate, odd parity, even parity or no parity. The service routines serialize and deserialize the characters at the requested rate, making use of the crystal-controlled real-time clock.

Parameters are passed to the EIA I/O routines in the form of a 4-byte EIA-control block. The address of the control block must be loaded into the H and L registers, or the index register for the 6800, prior to calling the send or receive subroutines.

8.4.1 EIA Send Routine

The send-control block (SCB) contains the fields shown in Table 8.3. These fields represent 4 sequential bytes in memory and provide the EIA send routine with the information necessary to process the request. The character to be transmitted must be passed in the accumulator (A) register, while the SCB address must be loaded into H and L, or index register, as noted above.

The first field of the SCB, the SCBFLAG field, contains option bits. The low-order bit of this field when set to 1 causes the send routine to generate a break signal, which is a 200 ms spacing level on the EIA line. When the break bit is set, no character transmission takes place, and control is returned to the calling program when the break has been transmitted.

Bits 1, 2, and 3 of the SCBFLAG field are used to select transmission options. Bit 1 selects two stop bits; normally when bit 1 is zero, only one stop bit is transmitted. Two stop bits are required when sending data to a Teletype. In other cases, two stop bits are allowed but are not required. Bit 2 of SCBFLAG, when set, inhibits generation of the parity bit. When bit 2 is

zero, parity is generated. Bit 3 is only meaningful when parity is being generated, and it specifies odd parity when set and even parity when zero.

The SCBRATE field of the EIA-control block contains a code which selects the transmission rate (Baud rate). Ten rate codes are allowed and are listed in Table 8.5. A rate code other than those listed in the table will have an unpredictable effect. The SCBLEN field specifies the number of bits in the data to be transmitted. Usual values are 5, 6, 7, or 8. Other character lengths will cause unpredictable results. Transmission of the byte passed in the accumulator begins with the low-order bit. Thus when 7 is specified as the length, the low-order 7 bits are transmitted.

The SCBUNIT field selects which EIA line is to be used. The value 1 selects EIA line 1 and the value 2 selects EIA line 2.

All registers and flags are saved and restored by the send routine.

Field	Bytes	Functions
SCBFLAG	1	Contains flag bits having the following meaning:
		Bit 0 --- The received data is shifted into the low-order bit of the accumulator. High-order unused bits will be zero.
		Bit 1 --- Disables parity bit generation.
		Bit 3 --- Selects odd parity.
		Bits 4-7 --- Unused
SCBRATE	1	Transmission rate select code (see Table 8.5).
SCBLEN	1	Length in bits of characters to be transmitted.
SCBUNIT	1	EIA line select (1 or 2).

Table 8.3 Send-Control Block

8.4.2 EIA Receive Routine

The receive-control block (RCB) contains the fields shown in Table 8.4. These fields represent four sequential bytes in memory and provide the EIA receive routine with the information necessary to process the request. The address of the RCB must be loaded into the H and L registers, or the index register for the 6800, prior to calling the routine. The received character will be in the accumulator (A) register upon return.

The first field of the RCB, the RCBFLAG field, contains option bits. For compatibility with the SCBFLAG field, only bits 2 and 3 are used in RCBFLAG. Bit 2, when set, disables the parity check. When bit 2 is zero, a parity check will be performed, which means that the parity bit will be read and the parity will be computed. If parity is incorrect, the error is indicated by setting the carry flag prior to returning to the calling program. If parity is correct, or if parity checking is suppressed, the carry flag will be set to zero upon return. The other bit of RCBFLAG which is used, bit 3, has meaning only when the parity check is enabled by bit 2 being zero. In this case, bit 3 selects odd parity when set and even parity when zero.

The RCBRATE, RCBLEN, and RCBUNIT fields have the same meaning as in the SCB. The RCBRATE field contains a code which selects the receiving Baud rate. Ten rate codes are allowed, and these are listed in Table 8.5. The RCBLEN field specifies the length, in bits, of the data to be received. Usual values are 5, 6, 7, or 8. The received data is shifted into the low-order bit of the accumulator. High-order unused bits will be zero. The RCBUNIT field selects the EIA line; either 1 or 2 is allowed.

All registers are saved and restored by the EIA receive routine except the accumulator and flags. The accumulator is used to return the received character and the carry flag is used to indicate a parity error.

Field	Bytes	Function
RCBFLAG	1	Contains flag bits having the following meaning:
		Bits 0-1 --- Unused
		Bit 2 --- Disable parity check
		Bit 3 --- Odd parity select
		Bits 4-7 --- Unused-should be zero
RCBRATE	1	Receive rate select code (see Table 8.5.).
RCBLEN	1	Character length in bits.
RCBUNIT	1	EIA line select (1 or 2).

Table 8.4 Receive Control Block

Code	Rate (Bits/Second)
0	50
1	75
2	110
3	134.5
4	150
5	300
6	600
7	900
8	1200
9	1800

Table 8.5 Baud Rate Select Codes

8.4.3 Setup for ASCII Data

The most common application for the EIA send/receive routines is for the exchange of ASCII character data. ASCII data has the following specifications:

Parity	Yes/Even
Stop Bits	Two for TTY, one otherwise
Length	7 bits

The control block setups for communication with a Teletype on EIA line 1 is as follows:

```
SCBFLAG = X'02'
SCBRATE = 2
SCBLEN  = 7
SCBUNIT = 1

RCBFLAG = X'00'
RCBRATE = 2
RCBLEN  = 7
RCBUNIT = 1
```

The control block setups for communications with a modem at 300 Baud on EIA line 2 is as follows:

```
SCBFLAG = X'00'
SCBRATE = 5
SCBLEN  = 7
SCBUNIT = 2

RCBFLAG = X'00'
RCBRATE = 5
RCBLEN  = 7
RCBUNIT = 2
```

9.0 Direct I/O Programming

While the I/O Service routines in the Monitor provide for high-level interfaces to most of the Microsystem peripheral devices, it may be necessary in some applications to access the devices directly. This section treats the direct programming of the standard I/O devices.

The I/O addresses are shown for the 8080A and Z-80 CPU's. For the 6800, it is necessary to map the I/O addresses into the memory space. All I/O devices (within the Microsystem) are located between X'D000' and X'DFFF'. Thus, the equivalent of an IN X'F0' command would be LDAA X'D0F0'. Similarly, the command OUT X'F3' would be STAA X'D0F3', for the 6800.

All of the memory reference instructions of the 6800 may be used with the I/O devices. However, the user should be very careful when using output-like commands (CLR for example). These commands may first generate a read cycle at the same I/O address. In some cases, the read I/O commands are used as reset signals. Note especially the commands IN X'F2', IN X'F3', IN X'E3' and IN X'E7' are used as status latch reset pulses. Thus a CLR command at the corresponding output addresses would also reset these status latches.

9.1 Keyboard Input

Asssociated with keyboard input is a status bit, interrupt mask bit and four I/O commands. The I/O commands are summarized in Table 9.0.

I/O CMD	FUNCTION	I/O BIT ASSIGNMENTS							
		7	6	5	4	3	2	1	0
IN X'F0'	Read data	HERE IS	Data 6	Data 5	Data 4	Data 3	Data 2	Data 1	Data 0
IN X'F1'	Read stat	-	-	-	-	-	-	-	stat
IN X'F2'	Reset stat	-	-	-	-	-	-	-	-
OUT X'F1'	Set mask	-	-	-	-	-	-	-	mask

Table 9.0 Keyboard Input Commands

The keyboard status bit is set to one to indicate that a key has been depressed. This status bit is read using the "read status" command (IN X'F1') which brings the status into the low-order bit of the accumulator. Once a key depression has been recognized by reading the status, the keyboard data can be read using the "read data" command (IN X'F0'). Also, the status bit must be reset by the the "reset status" command (IN X'F2'); the status remains set until cleared by this command. Since the data is only valid while

the key is depressed, data must be read from the keyboard within approximately 100 ms from the time that the status bit is first set.

The low-order seven bits of the data byte from the keyboard contain the ASCII encoding of the character that was typed. The HERE IS key, however, is connected directly to the high-order bit of the data byte. This key does not cause the keyboard status bit to be set and is not encoded as an ASCII character. When pressed, the HERE IS key is read as a zero bit, and when released it is read as a one bit. This allows the key to be used as a directly readable function or escape at the programmer's option.

The mask bit can be used to enable interrupts from the keyboard status bit. When the master system reset is generated at power-on or by the RESET button, the mask bit is cleared to zero and disables interrupts from the keyboard. If the the mask bit is set to a one by outputting an X'01' using the "set mask" command (OUT X'F1'), keyboard interrupts are then enabled. In order for the interrupts to be accepted by the CPU, it must be enabled as well using the "EI" instruction or the "CLI" instruction of the 6800. If both the the CPU and the keyboard interrupt masks are enabled, then when the keyboard status is set, an interrupt will be generated which causes the 8080A or Z-80 CPU to jump execution to location X'0028'. The 6800 will respond to a keyboard interrupt by a jump to the location stored in X'FFF8-9' (maskable interrupt). When processing an interrupt, the CPU automatically disables interrupts. Before the CPU interrupts are reenabled using the EI instruction, it is necessary that the cause of the interrupt (the keyboard status) be reset using the keyboard "reset status" command.

Keyboard interrupts are also disabled using the "set mask" command. Outputting an X'00' with the set mask command then disables further interrupts from the keyboard.

9.2 "BREAK" Key

The BREAK key also has an associated status bit, an interrupt mask, and three I/O commands as summarized in Table 9.1.

I/O CMD	FUNCTION	I/O BIT ASSIGNMENTS							
		7	6	5	4	3	2	1	0
IN X'F1'	Read stat	-	-	-	-	-	-	stat	-
IN X'F3'	Reset stat	-	-	-	-	-	-	-	-
OUT X'F2'	Set mask	-	-	-	-	-	-	mask	-

Table 9.1 Break Key Commands

The BREAK key has separate status and mask bits from the status and mask associated with other keys. The BREAK key is provided as an interrupt-generating escape key and is used by the system monitor/debugger for this function.

The break status bit is set to one when the BREAK key is depressed. This status bit read using the "read status" command (IN X'F1'), which brings the status bit into the accumulator in bit position 1. The status bit is reset back to zero using the "reset status" (IN X'F3') command.

The mask bit can be used to enable interrupts from the break status bit. The master system reset always clears the break mask bit, thus disabling interrupts form the BREAK key. If the mask bit is set to one by outputting an X'02' using the "set mask" command (OUT X'F2'), interrupts from the BREAK key are then enabled. The CPU interrupts must also be enabled by the "EI" instruction before the BREAK key interrupt will be recognized by the 8080A or Z-80 CPU. The BREAK key interrupt is vectored through the Non-Maskable Interrupt for the 6800. Thus the "SEI" and "CLI" commands will have no effect. When enabled, the break key status will produce an interrupt which causes the 8080A or Z-80 CPU to jump execution to location X'0020'. The BREAK key interrupt causes the 6800 to jump to the address stored in X'FFFC-D' (Non-Maskable Interrupt). The interrupt will remain pending until the status bit is reset by the reset command.

BREAK key interrupts are also disabled using the set mask command. Outputting an X'00' with the set mask command then disables further interrupts from the BREAK key.

9.3 CRT Display Select

The CRT display interface has the capability of refreshing the display from any 1K page of memory. The display page is selected using the following command:

I/O CMD	FUNCTION	I/O BIT ASSIGNMENTS							
		7	6	5	4	3	2	1	0
OUT X'F0'	Addr slct	Addr 15	Addr 14	Addr 13	Addr 12	Addr 11	Addr 10	-	-

Table 9.2 CRT Display Select

The CRT display address selection is accomplished by loading the accumulator with the high-order byte of the display page address and then executing the address select command (OUT X'F0').

It should be noted that the CRT display is always refreshed from a

page of memory beginning on an integral 1K boundary. The display is not capable of refreshing from an arbitrary 1K block of memory.

9.4 CRT Display Output

Since the CRT display is refreshed from the microcomputer memory, direct output is accomplished by simply storing data in the memory page selected for refresh. Data in this memory is converted by the CRT display module into a video signal which represents each ASCII code as a 5 x 7 dot matrix.

Since only 64 displayable characters are generated, only the low-order six bits of each byte are required for the display. The two high-order bits, therefore, may be arbitrarily set or reset without affecting the display.

The display is formatted from the CRT refresh memory as follows. The character at the top left of the screen is generated from the first byte. Following bytes correspond to sequential characters across the 40-character line. The next line is similarly generated from left to right using the next 40-characters. Table 9.3 shows the offsets from the beginning of the display page, corresponding to the first character of each line. As noted above, the CRT display can be selected to refresh from any 1K memory page.

The monitor/debugger and CRT output routine both set the display page to X'DC00'. For the 6800 the corresponding page begins at X'FC00'. If there is no reason to reselect the display page, this is the most convenient page to use since it corresponds to the last 1K page in memory.

Line	Offset	Line	Offset	Line	Offset
1	+X'0'	9	+X'140'	17	+X'280'
2	+X'28'	10	+X'168'	18	+X'2A8'
3	+X'50'	11	+X'190'	19	+X'2D0'
4	+X'78'	12	+X'1B8'	20	+X'2F8'
5	+X'A0'	13	+X'1E0'	21	+X'320'
6	+X'C8'	14	+X'208'	22	+X'348'
7	+X'F0'	15	+X'230'	23	+X'370'
8	+X'118'	16	+X'258'	24	+X'298'

Table 9.3 CRT Display Line Offsets

9.5 Memory Protect

Each 8K memory module contains an 8-bit protection register which can be read and reset by I/O commands. Each bit of this register indicates the write-protect status of a 1K page of the module.

The protection I/O commands are summarized in Tables 9.4 and 9.5.

I/O BIT ASSIGNMENTS

7	6	5	4	3	2	1	0
Page 7	Page 6	Page 5	Page 4	Page 3	Page 2	Page 1	Page 0

Table 9.4 Protect Command Bit Assignments

I/O CMD	FUNCTION	
IN X'1F'	read module 0 protection	(X'0000'-x'1FFF')
IN X'3F'	read module 1 protection	(X'2000'-X'3FFF')
IN X'5F'	read module 2 protection	(X'4000'-X'5FFF')
IN X'7F'	read module 3 protection	(X'6000'-X'7FFF')
IN X'9F'	read module 4 protection	(X'8000'-X'9FFF')
IN X'BF'	read module 5 protection	(X'A000'-X'BFFF')
IN X'DF'	read module 6 protection	(X'C000'-X'DFFF')
	6800	(X'E000'-X'FFFF')
OUT X'1F'	set module 0 protection	(X'0000'-X'1FFF')
OUT X'3F'	set module 1 protection	(X'2000'-X'3FFF')
OUT X'5F'	set module 2 protection	(X'4000'-X'5FFF')
OUT X'7F'	set module 3 protection	(X'6000'-X'7FFF')
OUT X'9F'	set module 4 protection	(X'8000'-X'9FFF')
OUT X'BF'	set module 5 protection	(X'A000'-X'BFFF')
OUT X'DF'	set module 6 protection	(X'C000'-X'DFFF')
	6800	(X'E000'-X'FFFF')

Table 9.5 Memory Protect Commands

The low-order bit of the protection register is assigned to the first 1K page of the module, the high-order bit is assigned to the last 1K page, and the other bits are assigned to respective 1K pages. A one bit in the protection register causes write request to be inhibited at any address in the corresponding 1K page. A zero bit in the protection register allows writes into the memory. The master system reset signal clears all protection registers to zero and thus unprotects all of memory.

Each memory module has a separate input command which reads the protection register and a separate output command which resets the protection register. These commands are listed in Table 9.5. The range of memory addresses controlled by each I/O command is also listed in this table. Module 7 cannot be implemented in the standard Microsystem because it is assigned to the bootstrap loader PROM.

9.6 Real-time Clock

The real-time clock has an associated overflow bit, interrupt mask

bit, and five I/O commands. The I/O commands are summarized in Table 9.6.

I/O CMD	FUNCTION	I/O BIT ASSIGNMENTS							
		7	6	5	4	3	2	1	0
IN X'E2'	Read clock	data 7	data 6	data 5	data 4	data 3	data 2	- 1	- 0
IN X'E3'	Read ovflw	ovfl	-	-	-	-	-	-	-
IN X'E1'	Reset ovflw	-	-	-	-	-	-	-	-
OUT X'E2'	Set clock	data 7	data 6	data 5	data 4	data 3	data 2	data 1	data 0
OUT X'E3'	Set mask	mask	EIA	EIA	EIA	-	-	-	-

Table 9.6 Real-Time Clock Commands

The value in the clock register is read using the "read clock" command (IN X'E2'). While the clock counter is actually 8 bits, only the high-order 6 bits are transferred to the accumulator by the read command. The low-order 2 bits are always read as ones.

The clock register is loadable using the set clock command (OUT X'E2') in this case the entire 8 bits of the accumulator are transferred into the clock counter register.

When the clock counter overflows, an overflow bit is set. This bit is read into the high-order bit of the accumulator with the "read overflow" command (X'E3'). The overflow bit will remain set until it is cleared by the "reset overflow" command (IN X'E1').

The low-order bit of the clock counter is incremented every 8.00 microseconds. Since the clock counter is 8 bits, this means that the clock will overflow every 2.048 milliseconds. The rate at which each of the clock bits is incremented is shown in Table 9.7.

Bit	Interval
0	8 us
1	16 us
2	32 us
3	64 us
4	128 us
5	256 us
6	512 us
7	1.024 ms
overflow	2.048 ms

Table 9.7 Clock Bit Timings

The clock mask bit can be used to enable interrupts from the clock overflow. The clock mask is always initially cleared by the master system reset which disables interrupts from the clock overflow bit. Interrupts are enabled from the overflow bit by setting the mask bit to one. This is accomplished by outputting an X'80' using the "set mask" command (OUT X'E3'). When the CPU interrupts are also enabled by the "EI" instruction, then the clock overflow interrupt will be recognized by the 8080A or Z-80. The vector location associated with the clock overflow interrupt is X'0010'. For the 6800 the clock interrupt is not maskable by the "SEI" instruction. The clock interrupt will cause a jump to the address stored in location X'FFFC-D'. When the interrupt is processed, the CPU is automatically disabled, but the interrupt will remain pending until the overflow bit is cleared by the "reset overflow" command.

It should be noted that the "set mask" command also controls some functions associated with EIA output. In particular, other bits of the command byte are used in setting levels on three EIA transmitt lines. Thus, care must be exercised when using the "set mask" command while EIA communication is being used.

9.7 EIA Communications

The I/O commands associated with the EIA communications lines are summarized in Table 9.8.

I/O CMD	FUNCTION	I/O BIT ASSIGNMENTS 7	6	5	4	3	2	1	0
IN X'E3'	Read EIA	-	EIA2 CTS	EIA2 RX	EIA1 RX	-	-	-	-
OUT X'E3'	Send EIA	mask	EIA2 RTS	EIA2 TX	EIA1 TX	-	-	-	-

Table 9.8 EIA Commands

Data levels from the EIA lines can be read using the "read EIA data" command (IN X'E3'). EIA 1 is configured for a terminal, while EIA 2 is configured for a modem. The received data from EIA 1 is available as bit 4, while the received data and clear-to-send from EIA 2 is available as bits 5 and 6 respectively. A marking signal on the data lines or an inactive signal on the control lines is read as one bit, while spacing signal on the data lines or an active signal on the control lines is read as a zero bit.

Data levels can be set on the EIA lines using the "set EIA data" command (OUT X'E3'). Transmitted data for EIA 1 is generated from bit 4 of the output byte, while transmitted data and request-to-send for EIA 2 is generated from bits 5 and 6 respectively. A marking signal on the data lines or an inactive signal on the control lines is generated by a zero bit, while a spacing signal or active signal is generated by a one bit. This is inverted form the way the EIA levels are read.

While there are no interrupts generated directly from the EIA lines, the strategy for the design of a general EIA line handler involves use of the real-time clocks to generate sampling interrupts.

It should also be noted that the high-order bit of the "set data" command byte controls the real-time-clock interrupt mask. When the clock is used along with the EIA line, some care must be taken not to change the interrupt mask when outputting EIA data. This is especially true in the case of the 6800 CPU where the clock interrupt is not disabled by the "SEI" instruction.

9.8 Parallel I/O Port

The parallel I/O port is a versatile 8-bit bidirectional interface which can be used to connect to external devices. It is also well suited for interprocessor communication. Associated with the port is a status bit, interrupt mask bit, and nine I/O commands. The I/O commands are summarized in Table 9.9.

Use of the 8-bit I/O port generally involves interconnection of an external device and selection of appropriate strapping options. For details with regard to the logic interface and strapping options, refer to the "FUTUREDATA Hardware Reference Manual." The functions of the parallel port commands will be elaborated here.

I/O CMD	FUNCTION	I/O BIT ASSIGNMENTS							
		7	6	5	4	3	2	1	0
IN X'E0'	Read stat	-	-	-	-	stat	-	-	-
IN X'E4'	Read data 1	data 7	data 6	data 5	data 4	data 3	data 2	data 1	data 0
IN X'E5'	Read data 2	data 7	data 6	data 5	data 4	data 3	data 2	data 1	data 0
IN X'E6'	Send strb	-	-	-	-	-	-	-	-
IN X'E7'	Reset stat	-	-	-	-	-	-	-	-
OUT X'E4'	Send strb 1	-	-	-	-	-	-	-	-
OUT X'E5'	Send strb 2	-	-	-	-	-	-	-	-
OUT X'E6'	Send data	data 7	data 6	data 5	data 4	data 3	data 2	data 1	data 0
OUT X'E7'	Set mask	-	-	-	-	-	-	-	mask

Table 9.9 Parallel Port I/O Commands

The parallel port has 8 data lines which can be used to transmit or receive data. Three additional control lines can be used for request/acknowledge signalling. One line is used to send a strobe to the external device, and a second line is used to receive and latch such a strobe in a status flip-flop which can be read by the processor. The third control line, which controls the direction of the data on the 8 lines, is not setable using I/O commands and must be handled by the external device.

Four additional control lines are available, two of which are strobed during the reading of data from the 8-bit lines, and two are general-purpose strobes which can be used for signalling that data is available.

Reading of data from the 8 lines is accomplished using the "read data" commands (IN X'E4' or IN X'E5'). During the read operation, a strobe signal is generated by each command on a separate control line. These signals can be used by the external device to determine that a read is in progress and differentiate between two different read requests.

Data is sent to the output register for transmission via the 8 data lnes using the "set data register" command (OUT X'E6'). The contents of the accumulator are transferred to the parallel port register. The data in this register is available on the 8 lines when the external device uses the direction-control line to switch the port from input mode to output mode. Two commands are provided for signalling the external device that data is available (OUT X'E4' and OUT X'E5'). These commands are like the read commands in that they generate strobe signals on separate control lines which can be used by an external device to determine that data is avialable in the output register. These strobe signals do

not have to be used exclusively for indicating availability of output data but can be assigned for any signalling task.

The request/acknowledge control lines have several associated commands. The "send strobe" command (IN X'E6') generates another strobe signal to the external device. While this signal can be assigned any meaning, the intent was that it be used to signal a request/acknowledgment. A request/acknowledgment strobe received through the port is latched in a status flip-flop. This status bit is read using the "read status" command (IN X'E0') and is placed in bit 3 of the accumulator. Once recognized, the status bit is reset to zero using the "reset status" command (IN X'E7').

The parallel port mask bit can be used to enable interrupts fron the status bit. The mask bit is always intially cleared by the master system reset, which disables interrupts from the parallel port status bit. Interrupts are enabled from the status bit by setting the mask bit to one. This is accomplished by outputting an X'01' using the "set mask" command (OUT X'E7'). When the CPU interrupt flag is also enabled by the EI instruction, then the parallel port interrupt will be recognized by the CPU. The vector location associated with the parallel port interrupt is X'0018' for the 8080A and Z-80. For the 6800 the parallel port interrupt is vectored through the maskable interrupt (X'FFF8'). When the interrupt is processed, the CPU is automatically disabled, but the interrupt will remain pending until the status bit is reset by the "reset status" command.

9.9 Summary of Interrupt Vectors

Table 9.10 lists all interrupt vectors used by the standard FUTUREDATA peripheral devices. Though these vectors are assigned to one device each, there is no restriction on having additional devices using the same interrupt priorities.

RST	8080A Z-80 Vector Addr	6800 Vector Addr	Function
0	X'0000'	X'FFFE'	System reset
2	X'0010'	X'FFFC'	Real-time-clock overflow
3	X'0018'	X'FFF8'	Parallel port status
4	X'0020'	X'FFFC'	Break key
5	X'0028'	X'FFF8'	Keyboard strobe

Table 9.10 Interrupt Vectors

10.0 Appendix

10.1 Configuring Memory

Memory module address strapping is accomplished by repositioning the jumper in the socket at the lower left of the memory module board. Seven positions are available, labeled 0 through 6. The following table shows the address range corresponding to the jumper position.

Jumper	Address Range 8080A, Z-80	Address Range 6800
0	0000-1FFF	0000-1FFF
1	2000-3FFF	2000-3FFF
2	4000-5FFF	4000-5FFF
3	6000-7FFF	6000-7FFF
4	8000-9FFF	8000-9FFF
5	A000-BFFF	A000-BFFF
6	C000-DFFF	E000-FFFF

The positions for the jumpers for various memory configurations are shown in figure 10.

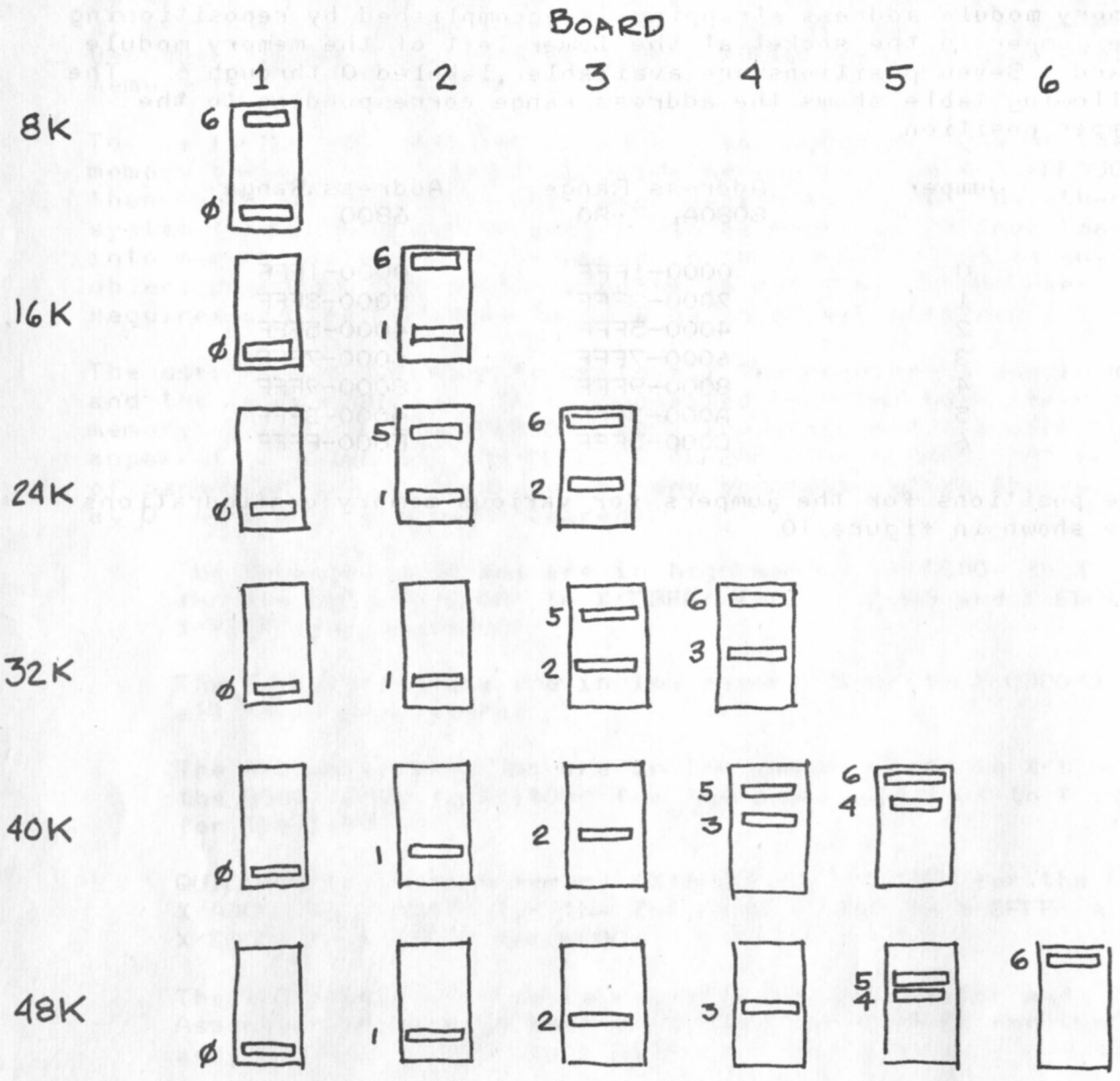

FIGURE 10. MEMORY STRAPPING

10.2 System Program Debugger

From time to time it is necessary for the user to modify the system software for his own purposes. When using the tape system or QUICKRUN it is most convenient to have a separate debugger located in unused memory to read, write and debug the software alterations. In order to facilitate this process a System Program Debugger "XDEBUG" has been written. XDEBUG runs in page 4 of memory (X'8000' to X'9FFF').

To use XDEBUG at least 16K of memory is required. One of the memory banks is strapped to provide memory in page 4. XDEBUG is then bootstrap loaded in the same fashion as any of the other system tapes. The system program to be modified is then loaded into memory using the "L" command in the same fashion as any other object program. The system program is modified as the user requires and then written to tape as an object program.

The user should remember to write all the required areas in memory and the entry address. It is suggested that the user clear all memory except for the XDEBUG area. The program areas will then be apparent. (note: Use the "LOAD", "BREAK", or "LOAD", "R" sequence of paragraph 4.2 to break out of any programs, since the vectors at 0 and X'38' have been cleared).

The Debugger programs are in high memory (X'CC00' to X'DBFF' for the 8080, X'C700' to X'DBFF' for the Z-80 and X'EB00' to X'FBFF' for the 6800).

The Editor programs are in low memory (X'0' to X'0D00') for all three processors).

The Assembler programs are in low memory (X'0' to X'1280' for the 8080, X'0' to X'1400' for the 6800, and X'0' to X'19C0' for the Z-80).

QUICKRUN is in high memory (X'A800' to X'DBFF' for the 8080, X'A000' to X'DBFF' for the Z-80 and X'A400 to X'BFFF' and X'E000' to X'FBFF' for 6800).

The I/O package, which is required for the Editor and Assembler is in high memory (X'D500' to X'DBFF' for the 8080 and Z-80 and X'F2C0' to X'FBFF' for the 6800).

The user should note the entry address for the various programs. This will appear in the "PC" field in the register display when the program is loaded. It should be the last parameter in the "W" command, if the entry is different from the first parameter specified.

10.3 Reassembling the Utility Program

The utility program is available in source form so that it can be modified by the user.

The utility program is modular in its organization. First, it is divided into two sections, one to handle source data transfers, and the other to handle object data transfers. Within each section are the routines which handle reading and those which handle writing. In the subsections which handle reading there are initialization routines for each I/O device, reading routines for each device, and termination routines for each device. Similarly, in the subsections that handle writing there are initialization, writing and termination routines for each I/O device.

Support for addtional I/O devices involves implementation of the appropriate read and/or write routines for source and/or object data.

In order to provide for transfers between different device types, the read and write routines transfer a minimum unit of data. For source data, this is one line. For object data, this one byte, with its intended memory address. The read routines are responsible for obtaining the data from the peripheral device, and deblocking if necessary into the minimum unit of data. The write routines are responsible for reblocking and outputting the data to the specified device.

Since the cassette tape routines are already implemented, other devices can transfer to and from the tapes by only adding the driver for the new device.

If a revised version of the utility is required it is prepared in the following manner. First the required changes are made in the utility source. It is then assembled and an object tape is obtained. Then XDEBUG (see 10.2) is used to configure a bootstrap loadable version of the new utility. The object tape of the modified utility is loaded with the "L" command. The bootstrap loadable version is now generated using the scatter write function in XDEBUG. The new utility must contain the revised utility and the monitor tape-I/O routines. These routines occupy the area from X'D500' to X'DBFF' (for the 8080A) or from X'F2C0' to X'fBFF' (for the 6800).

futuredata

11205 SOUTH LA CIENEGA BOULEVARD LOS ANGELES CALIFORNIA 90045 TWX 910-328-7202 TEL (213) 641-7700